Generación de historias centradas en el usuario en entornos 3D

Trabajo Fin de Grado en Ingeniería Informática

Alejandro Martín Guerrero
Frank Vito Julca Bedón

Dirigido por

Raquel Hervás Ballesteros
Gonzalo Méndez Pozo

Departamento de Ingeniería del Software
e Inteligencia Artificial
Facultad de Informática
Universidad Complutense de Madrid

Madrid, 15 de Junio de 2016

Documento maquetado con TEXIS v.1.0+.

Generación de historias centradas en el usuario en entornos 3D

Trabajo Fin de Grado

Alejandro Martín Guerrero
Frank Vito Julca Bedón

Dirigido por

Raquel Hervás Ballesteros
Gonzalo Méndez Pozo

Departamento de Ingeniería del Software
e Inteligencia Artificial
Facultad de Informática
Universidad Complutense de Madrid

Madrid, 15 de Junio de 2016

El verdadero conocimiento es saber
la magnitud de la propia ignorancia.
Confucio

Daría todo lo que sé por la mitad de lo que ignoro.
René Descartes

Agradecimientos

En primer lugar, dar las gracias por todas las veces que nos han soportado, escuchado y acompañado durante todo este año y que han permanecido de manera incondicional a nuestro lado. Gracias a las mesas de la primera planta donde, según hemos calculado, hemos estado más de 800 horas durante mañanas y tardes para sacar este proyecto.

Siendo ahora más serios, queremos agradecer a Gonzalo y Raquel, nuestros tutores, toda la ayuda y apoyo que nos han ofrecido. No solamente en las numerosas reuniones, sino también las veces que nos ayudaron a resolver las dudas que nos surgieron conforme íbamos avanzando en este trabajo de fin de grado. El buen ambiente, la afinidad y las risas han sido tres factores que siempre han estado presentes en cada reunión que hemos tenido, en momentos, incluso, llegando a molestar a algún profesor de al lado por el alboroto que organizábamos. En definitiva, muchas gracias, sobre todo, por motivarnos para dar todo lo que teníamos.

Además, queríamos mencionar al Departamento de Ingeniería del Software e Inteligencia Artificial por las veces que nos han prestado su ayuda en dudas puntuales sobre el entorno utilizado y otras dudas de la herramienta que ha hecho posible hacer esta memoria.

No nos olvidamos de nuestras familias, las cuales han estado apoyándonos en todo momento, dándonos ánimos y consejos que nos han servido como estímulo para seguir adelante, además de su paciencia y comprensión en los momentos más duros del proyecto. Por todo ello, gracias.

Por último, pero no menos importante, queremos agradecer a todos nuestros amigos esos momentos que nos han aportado para hacer que este último tramo de carrera sea más llevadero y más fácil de afrontar.

Autorización

Se autoriza a la Universidad Complutense de Madrid a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la memoria como el código, la documentación y/o el prototipo desarrollado.

Alejandro Martín Guerrero Frank Vito Julca Bedón

Resumen

En la actualidad, los videojuegos han ido adquiriendo cada vez más protagonismo en el sector de la tecnología, considerándose un arte para muchos y más que un entretenimiento para otros. Ha supuesto una de las principales fuentes de ingresos en las últimos años superando incluso al cine.

A día de hoy, hay muchas herramientas que permiten y facilitan su implementación. Concretamente encontramos programas software que ejercen de motores de desarrollo de videojuegos, como por ejemplo `Unreal Engine` o `Unity3D`, que brindan todo lo necesario para llevar a cabo esos juegos que tanto nos gustan.

Existen innumerables géneros, como los de aventura gráfica, que cobran más importancia porque intentan darle al usuario un mayor control intentando, cada vez más, simular la realidad. Por esta razón, surge el concepto de generación de historias en tiempo real, con el fin de diferenciarse de aquellas que están predefinidas y dar al usuario la posibilidad de crear una infinidad de historias que dependerán de los movimientos y decisiones que tome a lo largo del juego.

Este proyecto se centra en implementar esta idea, partiendo de un entorno gráfico que es el edificio de nuestra facultad. Sobre él desarrollaremos un sistema que permita al usuario moverse libremente por todo el edificio, generando distintas historias en función de los caminos que tome dentro de él. Existe la figura del narrador que le guiará y le aconsejará para completar los objetivos que vayan apareciendo. El usuario puede optar por hacerle caso y seguir sus indicaciones o ignorarlo.

Palabras clave

- Generación de historias
- Unity
- Narración
- Text-To-Speech
- Videojuego
- The Stanley Parable

Abstract

At present, video games have been acquiring an increasing leadership in the field of technology, being considered to be an art by many people and more than a mere entertainment by others. It has been one of the main sources of income in recent years, surpassing even the cinema industry.

Today, there are many tools that enable and facilitate their implementation. More specifically, we can find software programs which exert engine game development, such as `Unreal Engine` or `Unity3D`, which provide everything we need to develop the games we like.

There are countless genres, such as graphical adventure, which are becoming more important because they try to give the user more control trying, little by little, to simulate reality. For this reason, the need to generate stories in real time arises, in order to differentiate themselves from those that are predefined and give the user the ability to create an infinite number of stories that depend on the movements and decisions you make along the game.

This project focuses on implementing this idea, using a graphical environment which is the building of our Faculty of Computer Science. We will develop a system that allows the user to move freely around the building, creating different stories depending on the paths you take inside it. There is a narrator who will guide and advise you to complete the objectives that may appear. The user will be able to decid wether to follow his instructions or ignore them.

Keywords

- Story Generation
- Unity
- Narrative
- Text-To-Speech
- Videogames
- The Stanley Parable

Índice

Índice de figuras

Índice de Tablas

Capítulo 1

Introducción

Un camino de mil millas comienza con un paso.
Benjamin Franklin

1.1. Motivación

Desde la infancia, el ser humano intenta comprender todo lo que le rodea, siente interés por lo desconocido y busca descubrir cosas nuevas. A día de hoy, este sentimiento de aprender cada vez es mayor en un mundo donde la tecnología no para de evolucionar. Dentro de este sector, concretamente en el apartado de los videojuegos, la innovación es uno de los pilares imprescindibles sobre el que se sustenta. Surge, por ello, la necesidad de pensar en nuevas formas de asombrar al usuario y conseguir mantenerlo cautivado.

La generación de historias en tiempo real es una idea que desde hace unas décadas ha ido adquiriendo mayor fuerza en el mundo de los videojuegos. Desde las primeras aventuras gráficas, donde el usuario ya podía interactuar con personajes y objetos a través de un menú de acciones, hasta la actualidad, con aventuras conversacionales como *The Stanley Parable*, se ha intentado que el usuario cada vez fuese más partícipe de las historias, teniendo mayor libertad. Implica un progreso, en el que las historias ya no están totalmente predefinidas. Ahora, mientras juegas, dependiendo de las acciones realizadas y las decisiones tomadas pueden cambiar.

Este proyecto se centra en ese concepto. Desarrollamos un generador de historias, utilizando un motor de videojuegos, partiendo de un entorno gráfico que es el edificio de nuestra facultad. El jugador podrá moverse libremente por todo el edificio y se formarán distintas historias basadas en los movimientos que realice dentro de él. Nuestro personaje estará siempre acompañado

de un narrador que le ayudará a tomar decisiones y que le incitará a seguir unos determinados pasos. El usuario será libre de elegirlos o podrá ignorarlos si así lo desea. De este modo, podrá cambiar entre diferentes relatos e, incluso, llevar varios al mismo tiempo.

Mientras se implementaba este proyecto, se consideró la idea de hacer una estructura que no sólo cumpliera con lo anterior sino que además permitiera al usuario crear sus propias historias y enlazarlas según sus decisiones. Así, se decidió realizarla de una manera genérica para que cualquier persona fuese capaz de generlas, sin tener conocimientos de programación ni utilizar herramientas externas. Se integraron también elementos como personajes y objetos, que permitían llevar a cabo historias más dinámicas y reales. Por tanto, se consiguió que este trabajo se distinguiera de otros similares por aportar mecanismos necesarios para crearlas.

1.2. Objetivos

A continuación se describen los objetivos que se han planteado a lo largo de este proyecto.

- Generar dinámicamente historias dentro del entorno de la facultad dependiendo de las decisiones que tome el jugador.
- Crear un sistema de narración por voz que gestione y relate los mensajes de la historia en tiempo real, de manera que se puedan acumular para evitar la superposición de unos con otros. Además, la narración debe ser simultánea a la información que se muestre por pantalla.
- Establecer un comportamiento en el narrador que adopte distintos grados de intensidad en el relato de las historias, dependiendo de las acciones del usuario.
- Realizar un modo de edición que permita al usuario tomar parte de la construcción de las historias, mediante la obtención de diferentes magnitudes y modelización de objetos.
- Integración de personajes y objetos dentro de la facultad, en sitios específicos determinados por directivas proporcionadas por el usuario. Además, se distinguirá entre personajes que sirvan para dar una sensación más cercana a la realidad y otros que puedan interactuar e intervenir decisivamente en la historia.
- Añadir animación a los personajes, dotándoles de movimiento para que se puedan desplazar por la facultad sin necesidad de disponer del control del jugador. Además se establecerá un sistema de cambios de cámaras para focalizar dichos movimientos.

- Adaptar el proyecto de manera que funcione como una aplicación multiplataforma, para los sistemas operativos de *OS X* y *Windows*.

1.3. Estructura de la memoria

La estructura que hemos seguido para organizar esta memoria consta de los siguientes capítulos.

En los Capítulo 1 y 2 redactados en español e inglés respectivamente, se introduce el contexto en el que se realiza este proyecto, además de la motivación y los objetivos que se llevan a cabo. Por último, se explica la estructura de este documento.

En el Capítulo 3 se realiza, en primer lugar, una breve introducción a la narrativa que se ha conseguido desarrollar en los últimos años en el mundo de los videojuegos. Además, definimos algunos de ellos que marcaron un antes y un después en este género. En segundo lugar, nos detenemos para analizar el que ha servido como referencia de este proyecto, *The Stanley Parable*, y su relación con él.

En el Capítulo 4 se tratan una serie de conceptos relacionados con el entorno de `Unity`, que se han de explicar para entender gran parte de la implementación realizada. También se definen los algoritmos utilizados en el proyecto.

En el Capítulo 5 se detallan los métodos utilizados para calcular la localización del usuario, junto con los archivos que permiten la generación de las historias, y cómo se lleva a cabo la implementación del minijuego con el que puede interactuar el usuario.

En el Capítulo 6 se determina el desarrollo de las historias en función de los métodos de localización del usuario y cómo interactúan entre ellas, siguiendo las distintas plantillas para crearlas.

En el Capítulo 7 se describe el algoritmo de recorrido mínimo para la detección de desvíos dentro de la facultad, verificando su correcto funcionamiento mediante ejemplos de ejecución.

En el Capítulo 8 se habla del sistema de conversión de texto a voz que constituye el narrador y se define el comportamiento que ha de tener en función de los movimientos del jugador.

En el Capítulo 9 se establecen los tipos de personajes que conforman las historias, incluyendo el mecanismo necesario para su movimiento y el cambio de cámaras para centrar la atención del usuario en ellos.

En el Capítulo 10 se tratan los modos de juego del proyecto; uno dedicado a facilitar al usuario la edición de las historias y otro que le permita jugarlas.

En el Capítulo 11 se explican las aportaciones que hemos hecho cada uno de los autores del trabajo.

En los Capítulo 12 y 13 redactados en español e inglés respectivamente, se analizan las conclusiones y el trabajo futuro que hemos extraído de este trabajo de fin de grado.

Capítulo 2

Introduction

Los sueños parecen al principio
imposibles, luego improbables y luego,
cuando nos comprometemos, se vuelven
inevitables.
Christopher Reeve

2.1. Motivation

Since childhood, human beings try to understand everything around them, feel interest in the unknown and seek to discover new things. Today, this feeling of learning is increasing in a world where technology is constantly evolving. Within this sector, specifically in the field of video games, innovation is one of the essential pillars on which it is based. Therefore, the need to think of new ways to astonish the user and keep them captivated, arises.

Generating stories in real time is an idea that has been gaining force for some decades in the world of video games. Since the first graphic adventures, where the user could interact with characters and objects through a menu of actions, to date, with text adventures like *The Stanley Parable*, it has been tried the user to be more a participant and to have more freedom in the stories. It implies progress, in which the stories are not entirely predefined. Now, as you play, depending on the actions and decisions you make, these stories may change.

This project focuses on this concept. We have developed a story generator, using a game engine, based on a graphical environment which is the building of our faculty. The player can move freely throughout the building and different stories based on the movements he makes within it will be generated. Our character will always be accompanied by a narrator who will help him make decisions and entice him to follow specific steps. The user is free

to choose or ignore them if desired. Thus, he will be able to switch among different stories or even play several of them at the same time.

While this project was implemented, the idea of making a structure that not only meets the above but also allows the user to create their own stories and link them according to their decisions was considered. So we decided to do it in a generic way for anyone to be able to generate stories, without any programming knowledge or use of external tools. Elements such as characters and objects, which will allow to carry out more dynamic and real stories, were also integrated. Therefore, this work is achieved to be distinguished from other similar ones, for providing the needed mechanisms to create them.

2.2. Objectives

The objectives that have been raised throughout this project are as follows:

- Dynamically generate stories within the environment of the building depending on the decisions of the player.
- Create a system that manages narrative voice messages and tell the story in real time, so they can be accumulated and avoid overlapping with each other. In addition, the story should be simultaneous with the information displayed on the screen.
- Establish a narrator behavior which adopts varying degrees of intensity in the telling of the stories, depending on the user's actions.
- Perform an edit mode that allows the user to take part of the construction of the stories, by obtaining different magnitudes and modeling of objects.
- Integration of characters and objects within the faculty, at specific locations determined by guidelines provided by the user. There will be a distinction between those characters that will serve to give a feeling closer to reality and those others who may interact and intervene decisively in the story.
- Add animation to the characters, giving them movement so that they can move around the building without the need of player control. In addition, a camera changes system will be established to focus these movements.
- Adapt the project so that it will work as an application platform for operating systems *OS X* and *Windows*.

2.3. Document structure

The structure we have chosen to organize this memory consists of the following chapters.

In Chapters 1 and 2, written in Spanish and English respectively, we have introduced the context in which this project is done, plus the motivation and objectives carried out. Finally, the structure of this document is explained.

In Chapter 3 we provide a brief introduction to the world of video games narrative, which has been successfully developed in recent years. In addition, we have described some of them which, in our opinion, have marked a before and after in this genre. Secondly, we have stopped to analyze the one which has served as a reference for this project, *The Stanley Parable*, and its relationship with it.

In Chapter 4 we have discussed a number of concepts related to the environment `Unity`, which need to be explained to understand much of the implementation carried out. The algorithms used in this project are also explained.

In Chapter 5 the methods used to calculate the user's location are explained, along with the files that allow the generation of detailed stories. It is also explained how the implementation of the minigame the user can interact with is carried out.

In Chapter 6 the development of the stories based on the user location methods is determined and how they interact, following the various templates used to create them.

In Chapter 7 we have described the minimum path algorithm for detecting deviations within the faculty, verifying it works properly by working examples.

In Chapter 8 we describe the conversion system from text to speech, which is the narrator and the behavior that it must have, depending on the player's movements.

In Chapter 9 the types of characters that make up the stories are set, including the necessary mechanisms for their movements and changing of cameras to enable the user's attention to focus on them.

In Chapter 10 the game modes of the project are treated. We have differentiated them in two: one dedicated to provide the user the possibility to edit his own stories and an other one that allows him to play the stories.

In Chapter 11 we have explained the contributions of each group member.

In Chapters 12 and 13 written in Spanish and English respectively, the conclusions and future work we have drawn from this Final Degree Project are analyzed.

Capítulo 3

Trabajo relacionado

Enseñando aprendemos.
Séneca

3.1. Introducción

En este tema se tratan los aspectos que han servido como referencia para poder desarrollar este proyecto. En primer lugar nos centraremos en el juego *The Stanley Parable*, el cual ha supuesto el principal referente para llevar a cabo la forma de generar las historias.

Siendo la narrativa una de las partes más importantes dentro del género de aventura gráfica, es necesario introducir los principales conceptos que tiene, así como un análisis entre los juegos más influyentes dentro de este campo.

3.2. La narrativa en el mundo de los videojuegos

Se dice que saber transmitir mediante el uso de la palabra lo que piensas, lo que lees y lo que vives, en definitiva narrar, es un arte. Expresarse muchas veces es complicado, como diría Michel Eyquem de Montaigne: *"La palabra es mitad de quien la pronuncia, mitad de quien la escucha"*. Intentar llegar al sentimiento a través de la oración es algo que desde tiempos remotos ha sido complicado, desde los juglares que con sus cantares narraban las historias que en el futuro serían recordadas, hasta el día de hoy donde los guionistas son los que se encargan de escribir las palabras a las que más tarde los actores darán vida. Encontramos un mundo cuyo espectro es infinito, el cual se ha empezado a explotar no hace mucho en los videojuegos.

3.2.1. Introducción a la narrativa

La narrativa es un género literario que se basa en la narración de una serie de hechos, donde se ven involucrados uno o más personajes (Wikipedia, Narrativa). La comunicación es una de las partes más importantes a la hora de tener una buena interacción con el usuario y como vamos a ver se pueden dar diferentes clases de narrativa.

En primer lugar, es necesario hacer una distinción entre la narrativa lineal y la narrativa interactiva. La narrativa lineal está representada por historias cerradas, preestablecidas y en todo momento dependientes del narrador (Planells, 2010). En este caso solemos encontrar la estructura : inicio, nudo y desenlace.

A diferencia del modelo anterior, la narrativa interactiva se encuentra definida por relatos que pueden variar, dando lugar a historias abiertas. Pueden existir diversos inicios, así como también múltiples finales, y no existen un principio y fin preestablecidos. En cualquier caso, es el usuario el encargado de seleccionar uno de estos modelos narrativos para el desarrollo del relato, introduciendo de este modo la participación activa del usuario en la construcción de la narración.

La decisión de qué tipo de narrativa elegir, en el mundo de los videojuegos, se da principalmente en los de género de aventuras. En este tipo, el diseñador establece la trama narrativa que considere apropiada y aquellos elementos que permitan la cohesión del relato. Sin embargo, si únicamente considera estos factores y no permite un mínimo de libertad al jugador para tomar determinadas decisiones que él considere importantes, puede llegar a verse frustrado.

Es necesario que el jugador tenga cierto control sobre el juego o que al menos lo parezca. El narrador, por su parte, debe cederle más protagonismo, dejándole decidir qué objeto desea recoger o, incluso, el camino que quiere tomar. No obstante, no se debe prescindir de él, ya que en definitiva, el jugador sólo está resolviendo una historia y no creándola. Por tanto, es el narrador el encargado de redirigirle y limitarle dentro de lo permitido. Un ejemplo actual de censura y control absoluto se puede ver en las *cut scenes* (Wolf, 2012). En ellas el narrador cuenta hechos importantes para el desarrollo de la historia mediante secuencias no interactivas. Son muy habituales en la gran mayoría de juegos de aventuras y se utilizan normalmente para cambiar de escenarios o mostrar consecuencias de las acciones del jugador. En estos casos, el desarrollador decide que el narrador es el encargado de controlar ese fragmento narrativo y no el jugador, durante un breve periodo de tiempo.

La relación que se establece entre el jugador y el narrador puede variar, llegando incluso algunas veces a entremezclarse como pasa en muchos juegos

de primera persona donde el punto de vista es subjetivo (*Half-life*) (Valve, 2004) y por tanto no se define claramente el rol del narrador. Por el contrario, en otros juegos como puede ser *King's Quest* (Williams, 1984), el narrador se diferencia del jugador contándonos la historia en la que se ve involucrado el personaje.

3.2.2. King's Quest

Es un juego que narra la vida de Sir Graham, un caballero que tiene que recuperar tres objetos mágicos tras ser llamado por el rey: un espejo, un escudo y un cofre mágico que siempre está lleno de oro. A cambio de conseguir estos tres objetos el rey le obsequiará con la corona del reino.

En el aspecto de la interacción, el jugador tiene libertad para decidir hacia qué zona encaminarse, podrá realizar distintas acciones mediante la línea de comandos. Hay una gran cantidad de acciones y combinaciones que se pueden utilizar durante el desarrollo de la historia. El narrador adopta una postura neutra con un estilo meramente descriptivo ya que no es el personaje y, por tanto, no se implica en nada.

En cuanto al análisis del jugador, principalmente se aprecia que es más autónomo que en la mayoría de juegos de aquel entonces. Si ordena realizar una determinada acción al personaje, el narrador también asume esa decisión y se adapta a una nueva situación. Sin embargo, el personaje se limita a realizar las acciones haciendo de mediador entre narrador y jugador.

3.2.3. Monkey Island II

En esta segunda entrega de la saga Guybrush Threepwood es un pirata en busca del mayor tesoro del Caribe. Para llegar hasta él deberá enfrentarse al pirata zombi LeChuck mientras que va en busca de su amada Elaine Marley.

Monkey Island II (Salter, 2014) está comprendido por una interfaz dividida en dos partes. En la parte superior transcurre la historia, es decir, el entorno gráfico, mientras que la inferior está subdividida en dos zonas, una que contiene las acciones que puede realizar el jugador (tirar, andar, coger...) y otra el inventario de los objetos que lleva. Estas dos zonas constituyen la parte interactiva donde se combinan las acciones con objetos de inventario, o bien con los elementos jugables de la parte superior. Las acciones que puede realizar el jugador están predefinidas por el diseñador a únicamente nueve posibles, lo que limita en gran medida el sistema de interacción.

Analizando la narrativa del juego, se distingue rápidamente que no estamos ante un narrador que cuenta la historia desde una perspectiva ajena a la del

personaje como sucede en *The Stanley Parable* (Regueiro, 2015), sino que, al igual que juegos más actuales como *The Last of Us* (Minkoff, 2013), el narrador es el propio personaje y cuenta la historia desde el punto de vista del mismo.

En *King's Quest* el personaje es un tanto equidistante, aquí es claramente cercano al narrador. Lo identificamos como el protagonista, él cual nos está contando sus vivencias.

3.2.4. Conclusión

A medida que avanza la evolución de este género, la autonomía del personaje tiende a ser mayor, sin imponerle tantas restricciones. Este amplio sistema crea muchas más libertades para la interacción del jugador. Sin embargo, es necesario también disponer de un conjunto de restricciones que faciliten la unicidad del relato, aunque esto suponga que el jugador se vea frustrado al perder el control sobre sus acciones. Existe pues el dilema entre sí aplicar un sistema que ofrezca más control al jugador sobre el desarrollo de la historia o uno que limite sus opciones a cambio de que el diseñador pueda contar las historias que desee.

Al principio, durante el periodo de las aventuras de texto, era habitual que el protagonista tuviese una relación muy directa con el jugador y se confundiese con él. Esto permitía involucrarle mucho más, pues daba la sensación de la inexistencia del personaje. Por tanto, las aventuras de texto brindaban una mayor libertad al jugador en la toma de decisiones. Sin embargo, en el género sucesor de tipo aventura gráfica, para resolver este dilema tiende a usar un personaje más cercano al narrador para adquirir más control sobre el desarrollo de la historia.

En nuestro caso, la narración es una mezcla de las dos anteriores. El personaje tiene una gran relevancia en el desarrollo de la historia, puesto que tiene total libertad para moverse por toda la facultad, siendo aún más cercano al jugador que en *Monkey Island II*. La figura del protagonista es absorbida por el jugador. A su vez, como sucede en *The Stanley Parable*, el narrador le indica un camino a seguir intentando llevar el control de la situación y adquiere una mayor personificación, llegando a convertirse en un personaje más del juego.

3.3. The Stanley Parable

The Stanley Parable es un videojuego cuyo género se puede clasificar en una aventura conversacional (Lebowitz, 2012). Es un juego que transcurre en pri-

mera persona, donde no hay secuencias de acción o combates, simplemente el jugador controla al protagonista, Stanley, llevándolo a través de un ambiente surrealista, y narrando por voz las acciones que va realizando.

El desarrollador Davey Wreden concibió el juego después de considerar que la mayor parte de los videojuegos confinaban a los usuarios a sus reglas. La narración que se llevó a cabo planteó temas como la naturaleza de las elecciones y las predicciones dentro de los videojuegos.

El jugador tiene la oportunidad de tomar numerosas decisiones sobre qué caminos seguir, incluyendo decisiones que contradicen las indicaciones del narrador, produciendo diferentes narraciones y finales. El concepto que se pretende seguir es la generación en tiempo real de historias que permita narrar los pasos que ha seguido el usuario.

Durante el desarrollo de la historia, el jugador tiene libertad de movimiento y puede interactuar con los elementos que se encuentran a su alrededor, como presionar botones o abrir puertas, pero no tiene otros controles. La voz de un narrador cuenta la historia y empieza explicándonos que el protagonista Stanley trabaja en unas oficinas monitorizando datos en su pantalla del ordenador y presiona botones sin que nadie le cuestione nada. De repente, un día la pantalla se apaga y Stanley, indeciso, comienza una exploración por el edificio a medida que se da cuenta de que está solo.

Llegados a este punto empieza realmente el desarrollo de la historia, la cual se puede dividir en varias posibilidades que estarán relacionadas con las elecciones que haga el jugador. En el momento en el que llega a un punto de la historia donde se tiene que plantear qué camino tomar, el narrador interviene sugiriendo uno a Stanley para avanzar. El jugador, en su infinita capacidad de elección, puede optar por contradecir al narrador, obligando a que readapte el diálogo a una nueva situación si el jugador persiste en ignorarlo, o hacer caso a sus sugerencias, siguiendo así una historia predefinida. En el juego existen seis posibles finales en la modificación inicial. Si se quisiera experimentar con todos los caminos posibles, está diseñado para que tenga una duración aproximada de una hora.

Wreden, el desarrollador, era alguien sin experiencia en el motor de videojuegos Source sobre el que está realizado *The Stanley Parable*. Se basó en la información y ayuda obtenida por foros sobre el Source Development Kit, enseñándose a sí mismo los principios básicos. El juego fue creado completamente por él salvo la voz del narrador, que fue interpretada por un actor. Aunque la intención inicial del desarrollador Wreden fue un proyecto personal para tratar de hacer un juego que planteara las preguntas sobre por qué las personas juegan a los videojuegos, descubrió que otros jugadores habían estado considerando los mismos tipos de preguntas. Nosotros, al igual que Wreden, fuimos autodidactas y nuestra formación se adquirió gracias a

videotutoriales, foros y el método convencional de ensayo y error.

Este concepto de generar las historias es el que se desea realizar en el ámbito de nuestra facultad, mediante distintas historias narradas y diferentes caminos que harán al jugador enfrentarse a distintos retos incluyendo minijuegos, personajes y objetos que intervendrán decisivamente simulando este juego. De esta forma y a diferencia de *The Stanley Parable*, en este proyecto desarrollamos una herramienta para que otros usuarios puedan crear sus propias historias, que se generan en tiempo real a la vez que se narran las decisiones que toma el jugador sugiriéndole posibles caminos.

Capítulo 4

Conceptos previos

Me lo contaron y lo olvidé; lo vi y lo entendí; lo hice y lo aprendí.
Confucio

4.1. Introducción

Introduciremos el motor de videojuegos que ha hecho posible este trabajo de fin de grado y los elementos más importantes que se utilizarán. Además haremos hincapié en los principales aspectos sobre los algoritmos planteados y utilizados para conseguir que sea lo más real posible de una forma óptima.

4.2. Motor de videojuegos Unity

Es necesario explicar algunos conceptos básicos para poder entender los detalles de implementación de la aplicación. A través de la información proporcionada por la página oficial de Unity3d (Technologies, 2005), conseguimos adquirir la mayor parte de estos conocimientos. Partimos con este motor de videojuegos multiplataforma que es capaz de simular entornos 3D debido a que el proyecto se desarrolla en él. A causa de que en proyectores anteriores con una temática parecida ha dado buenos resultados, se ha propuesto continuar utilizándolo. Los lenguajes soportados por el entorno y utilizados en el proyecto son: JavaScript, C y XML.

A continuación se determinan los principios básicos para la utilización de Unity3D:

- `Scene`: representa una colección de objetos del juego configurados para representar el ambiente, obstáculos y el comportamiento. Se debe tener

un proyecto organizado en varias escenas para trabajar correctamente.

- `Collider`: es un componente que permite determinar colisiones físicas entre un objeto y el mismo. Es invisible durante el juego y no tiene por qué ser de la misma forma que el objeto al que está asociado.
- `Trigger`: son eventos asociados a objetos del juego que se activan cuando se cumplen determinadas acciones.
- `GameObject`: son los objetos del juego más importantes de Unity también denominados contenedores. Necesitan propiedades especiales para poder interactuar con ellos. Algunos ejemplos son: Cubes, Sphere, Capsule, Plane, Terrain o Tree.
- `Script`: es el archivo que determina el comportamiento que tiene un objeto del juego al que está asociado. Es posible añadir mas de un `script` a un mismo objeto.
- `Prefab Asset`: actúa como una plantilla a partir de la cual se pueden crear nuevas instancias del objeto en la escena. Cualquier edición hecha a un prefab asset será inmediatamente reflejada en todas las instancias producidas por él.
- `Mesh Renderer`: es el filtro que se agrega automáticamente al objeto y permite representarlo.
- `Mesh Collider`: es un componente que se utiliza para importar datos en formato Maya y también sirve para detectar colisiones con el entorno.

4.2.1. Base de derivación de todos los scripts

`Monobehaviour` es la clase base de la que derivan todos los `scripts`. Contiene métodos que pueden ser ejecutados en determinados momentos del desarrollo del juego. Algunos de los más importantes son los siguientes:

- `void Awake()`: se llama solamente una vez por cada objeto del juego. Sirve para resolver referencias entre `scripts` y se ejecuta antes que la función `Start()`.
- `void Start()`: es invocado una vez durante el tiempo de vida del script en el que se encuentre, justo en el momento en el que el `script` se ha activado y antes de la primera invocación del método `Update()`. Se usa principalmente para inicializar los elementos del juego y establecer la configuración inicial en tiempo de ejecución.
- `void Update()`: es invocado en cada fotograma si el `Gameobject` en el que se encuentra el `script` que contiene dicha función está activa-

do. Es la función más utilizada para implementar cualquier tipo de comportamiento del juego que requiere una comprobación constante.

- `void OnGUI()`: se usa para la representación y el manejo de eventos `GUI`. Es llamado múltiples veces durante cada frame del juego, si el `Gameobject` en el que se encuentra el `script` que contiene dicha función está activado.
- `void OntriggerEnter()`: función que se activa cuando un objeto colisionador entra en contacto con otro objeto. Función análoga a la de `OnTriggerExit()` pero cuando ambos objetos dejan de colisionar.
- `void OnDestroy()`: es una función que permite realizar una acción cuando se elimina la clase MonoBehaviour. Sólo se llamará cuando el objeto asociado, esté activado.

4.2.2. Recursos estándar

Unity ofrece `Assets Packages`[1] que contienen múltiples `Standard Assets`[2] con los que se puede implementar el comportamiento y modelizar los objetos del juego. Este conjunto de `assets` son bastante utilizados por la mayoría de sus clientes, puesto que permiten controlar las funciones básicas de un personaje, la infraestructura del entorno en el que se mueve el jugador o incluso la distribución de las cámaras del juego. Aunque en algunos casos esta colección no sea suficiente para obtener el resultado deseado, como sucede en este proyecto, *Unity* facilita la posibilidad de importar paquetes personalizados para cumplir dicho objetivo.

Uno de los paquetes más importantes del proyecto es `New Character Pack` que incluye modelos humanoides de personajes con diversas animaciones y un gran conjunto de complementos y texturas. Aunque no dispone de modelos de `scripts` de comportamiento para los personajes, sí contiene personajes personalizables y con una apariencia humana más cercana a la realidad que los personajes estándar de *Unity*.

4.2.3. Serialización

Esta clase permite controlar el modo en que se codifican los objetos, de forma que pueden ser convertidos en documentos XML(serialización). Más tarde se pueden construir de nuevo a partir de dichos documentos (deserialización).

[1] Son una forma práctica de compartir y reutilizar paquetes de otros proyectos de Unity.
[2] Son los paquetes estándar que por defecto tiene Unity.

La serialización es un proceso de estado de almacenamiento de un objeto. El primer paso es marcar varios elementos utilizando atributos de serialización. En la Tabla 4.1 se presentan los atributos principales utilizados:

Atributos	Descripción
[XmlRoot]	Elemento raíz: Este atributo representa el elemento raíz del documento XML resultante. Sólo puede haber un elemento raíz en un documento XML.
[XmlAttribute]	Atributo: Este atributo representa la propiedad de persistir como atributo XML en el documento XML resultante.
[XmlElement]	Elemento: Este atributo marca una propiedad como un elemento XML. La propiedad Elemento del atributo se utiliza para establecer el nombre del elemento.
[XmlArray]	Array: Este atributo indica la representación de un array de elementos en un documento XML.
[XmlArrayItem]	Elemento de array: Este atributo indica la representación del tipo de elemento de un array en un documento XML.

Tabla 4.1: Atributos de archivos XML

4.2.4. Escenas

Las escenas contienen todos los objetos que va a tener cada nivel de nuestro juego. Se usan para crear menús principales, niveles individuales, y cualquier otra cosa en donde colocar un determinado entorno con diversos obstáculos, y/o decoraciones.

Si la aplicación contiene más de una escena, es imprescindible establecer un mecanismo para cambiar entre ellas, que permita conservar los datos del progreso del jugador según avance en el juego. La dificultad que surge cuando se carga una nueva radica en que todos los objetos y datos de la escena anterior se destruyen y no se instancian en ésta. Para conservar los más importantes y que no constituyen necesariamente objetos del juego, como el identificador de la escena actual, se hace uso de *variables estáticas* que permanecen durante toda la ejecución de la aplicación, sin depender de la escena en la que se encuentre el jugador.

La utilización de *variables estáticas*, como método de intercambio de datos,

no es suficiente si se trata de mantener una mayor cantidad de datos que suponen un coste de carga en memoria elevado (como sucede en este proyecto), por lo que se tienen que almacenar en forma de estructuras que residen en los objetos e indicar que éstos no se van a destruir.

El cambio de una escena a otra se realiza mediante `Application.LoadLevel (NameScene)`.

4.2.5. Canvas

`Canvas` es un espacio abstracto que representa el componente raíz que contiene todos los elementos *User Interface*(UI) que existen en el juego. Aunque no es común, también es posible usar más de un `Canvas` en la misma escena, incluso anidarlos para que uno esté dentro de otro. Tanto si se aplica como si no, todos los componentes de cada `Canvas` se dibujan en el mismo orden en el que aparecen en su jerarquía. Si dos o más elementos se superponen, entonces el último de ellos aparecerá encima del resto.

Cabe destacar que en este proyecto los `Canvas` forman, principalmente, los menús que van a permitir al jugador cambiar de escenas que se detallarán en capítulos posteriores. Así pues, el modo de renderización adecuado será `Space Screen Overlay` porque nos interesa, por ejemplo, que si el jugador accede al menú de pausa para salir del juego, todos los objetos de la escena pasen a segundo plano para visualizarlo correctamente.

4.2.5.1. Renderización de los elementos

Tradicionalmente, la forma en la que se presentaban los elementos `UI` es a través de un diseño gráfico en *2D* dibujado directamente en la pantalla. Para obtener mejores resultados se necesitaban procesos que generaran diseños con imágenes en *3D* como la renderización.

`Renderizar` es un término que se usa para referirse a la técnica que pretende imitar un entorno tridimensional que de otra forma requeriría una potencia de cálculo demasiado elevada.

`Unity` soporta varios tipos de renderización del espacio de la pantalla en los que aplica un espacio en *3D* visto por una cámara. También permite que los elementos sean renderizados como objetos en la escena. Los modos disponibles dependen de si utilizan espacio de la pantalla o del propio juego para organizar los componentes.

- `Ocupar el espacio de la pantalla (Screen Space)`: en este tipo se utiliza toda su extensión de manera que el `Canvas` es escalado para encajar con ella. Aunque la resolución cambie, siempre se reajustará

automáticamente para ocupar el máximo espacio posible. Si se usa el modo `Overlay` (ver Figura 4.1), los elementos `UI` aparecerán superpuestos a cualquier objeto de la escena, mientras que en el modo `Camera` puede que no ocurra, como se puede ver en la Figura 4.2 En este último, el `Canvas` se dibuja como si fuera un plano que se encuentra de frente a la pantalla y a una distancia determinada. Por eso los objetos del juego pueden atravesarlo.

- `Ocupar el espacio de la escena (World Space)`: El `Canvas` es un objeto plano que puede ser traspasado por objetos de la escena, como se ve en la Figura 4.3.

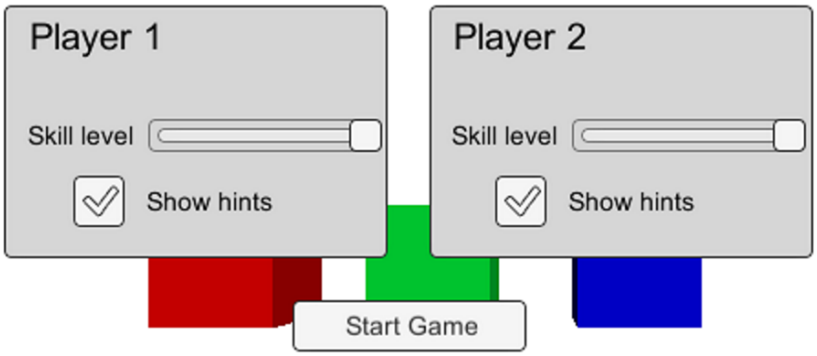


Figura 4.1: Modo Overlay

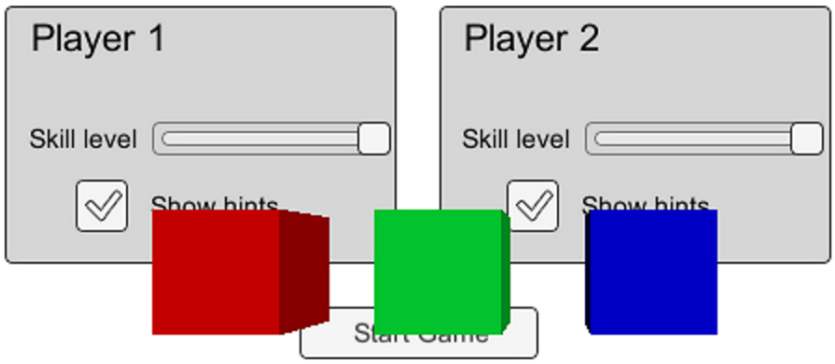


Figura 4.2: Modo Camera

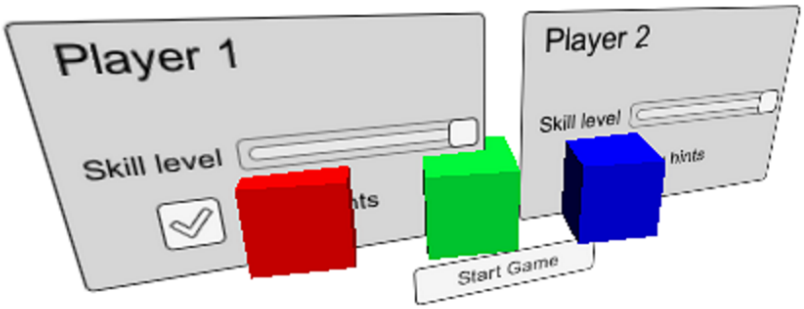


Figura 4.3: Modo World Space

4.3. Algoritmos de búsqueda en grafos

Como se verá más adelante, para hallar el recorrido mínimo desde una posición a un objetivo se utilizan algoritmos de búsqueda del camino más corto (Heileman, 1996). Su principal uso se da en la teoría de grafos y se adecuan perfectamente a esta parte del proyecto.

El problema consiste en encontrar un camino entre dos nodos de tal manera que la suma de los costes de las aristas que lo constituyen sea mínima. Un ejemplo de esto es encontrar el camino más rápido para ir de un aeropuerto a casa después de volver de un viaje. En este caso, el primer nodo representaría el aeropuerto y el segundo la casa, las aristas las carreteras que las unen y el coste viene dado por el tiempo que se emplea en atravesarlas.

Los algoritmos que se usan para solucionar este problema se clasifican en tres tipos:

- `Caminos más cortos desde el origen`, en el cual tenemos que encontrar los caminos más cortos de un nodo origen a todos los demás nodos del grafo.
- `Caminos más cortos desde el destino`, en el cual tenemos que encontrar los caminos más cortos desde todos los nodos del grafo a un único vértice destino.
- `Caminos más cortos entre todos los pares de nodos`, el cual tenemos que encontrar los caminos más cortos entre cada par de vértices en el grafo.

La mayoría de los algoritmos de búsqueda en grafos son perfectamente compatibles para cumplir esta función en nuestro modelo. Los tres algoritmos

que son afines al proyecto y que permiten encontrar el camino mínimo de forma óptima son los siguientes.

4.3.1. Algoritmo A* (A estrella)

Este algoritmo encuentra el camino de menor coste entre un nodo origen y un nodo destino siempre y cuando se cumplan unas determinadas condiciones. La complejidad computacional en el caso peor será exponencial, mientras que en el caso mejor, el algoritmo se ejecutará en tiempo lineal.

El espacio requerido por A* para ser ejecutado es su mayor problema. Dado que tiene que almacenar todos los posibles siguientes nodos de cada estado, la cantidad de memoria que requerirá será exponencial con respecto al tamaño del problema.

4.3.2. Algoritmo de Bellman-Ford

El algoritmo de *Bellman-Ford* determina la ruta más corta desde un nodo origen hacia los demás nodos, para ello es requerido como entrada un grafo cuyas aristas posean costes. La diferencia de este algoritmo con los demás, es que los costes pueden tener valores negativos ya que *Bellman-Ford* permite detectar la existencia de un ciclo negativo.

El algotitmo tiene un coste equivalente a $O(|V| \cdot |E|)$, donde $|V|$ es el número de vértices y $|E|$ el número de aristas respectivamente.

4.3.3. Algoritmo de Dijkstra

Es un algoritmo que sirve para determinar el camino más corto dado un vértice origen al resto de los vértices en un grafo con los respectivos costes de sus aristas.

La idea de este algoritmo consiste en ir explorando todos los caminos más cortos que parten del nodo origen y que llevan a todos los demás vértices. Cuando se obtiene el camino más corto desde el vértice origen, al resto de vértices que componen el grafo, el algoritmo se termina.

El orden de complejidad del algoritmo es $O(|V|^2 + |A|) = O(|V|^2)$ sin utilizar cola de prioridad, $O((|V| + |A|)log|V|) = O(|A|log|V|)$ utilizando cola de prioridad (por ejemplo un montículo).

Capítulo 5

Diseño de la aplicación

Sólo aquellos que se arriesgan a ir
demasiado lejos, pueden descubrir hasta
dónde se puede llegar.
T.S. Eliot

5.1. Introducción

En este capítulo veremos los primeros pasos que se llevaron a cabo para la implementación del proyecto. Concretaremos sobre los métodos utilizados para la localización del usuario y los archivos necesarios para la inicialización de la aplicación.

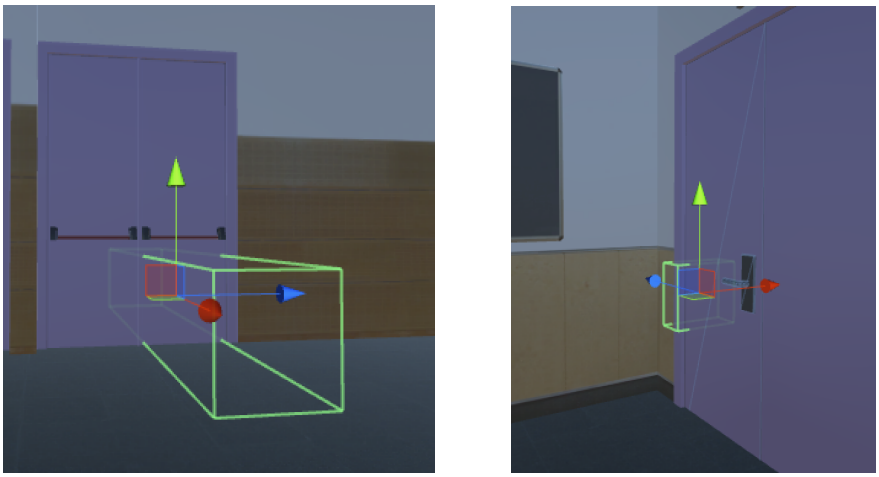
5.2. Materialización del edificio

Es el proceso con el cual conseguimos que el usuario pueda interactuar con el edificio, haciendo que éste sea tangible. Al importar el modelado 3D del edificio completo, descubrimos que se encontraba en formato MAYA (`.fbx`) por lo que al insertar un objeto en su interior, este caía de forma libre indefinidamente ya que prescindía de la parte material por lo que el usuario podía atravesar la infraestructura. Mediante el uso de `Mesh colliders` se resuelve este problema dotando de integridad física al edificio, los cuales ejercen la función de una malla recubridora rígida que se adapta a todos los objetos(paredes, mesas, sillas, etc.).

5.3. Recorrido del escenario

El desplazamiento por el interior de la facultad se vio afectado por el hecho de que todas las puertas estaban cerradas y, por tanto, el usuario no podía acceder a una habitación ni salir de ella. Para solucionarlo se llevó a cabo un estudio de las diferentes posibilidades para la resolución de este error. Finalmente, optamos por la implementación de dos tipos de `scripts` en *javascript* que definen el comportamiento de apertura de las puertas. El proceso consiste en un mecanismo de sistemas de puertas automáticas como se puede ver en la Figura 5.1, gracias a un engranaje que es el resultado de unificar dos objetos (`Cubes`) que están representados por una bisagra y un sensor de activación del mecanismo.

- `Bisagra`: gracias a uno de los `scripts`[1], realiza la rotación de la puerta.
- `Activador`: es un `trigger` que nos permite saber si otro objeto ha colisionado con este.



(a) Activador de la puerta (b) Bisagra de la puerta

Figura 5.1: Mecanismo de las puertas

5.4. Métodos de localización del usuario

Tras dar solución al problema de la interacción del usuario a través de la facultad, nos planteamos la siguiente pregunta: ¿Cómo determinamos la posición del usuario dentro del juego? Esta cuestión nos surge de la necesidad

[1] Debido a la posición de determinadas puertas hubo que invertir la rotación de estas en algunos casos.

de generar historias y diálogos dando una retroalimentación de la posición que ocupa, consiguiendo de esta forma saber los pasos que va a seguir en el transcurso del juego.

Después de determinar el objetivo, nos surgió una nueva pregunta: ¿Cómo llevar a cabo dicha localización? En los apartados siguientes se desarrollan las posibles soluciones.

5.4.1. Localización por recubrimiento de superficie

Este fue el primer método ideado para minimizar el coste y aumentar la eficiencia de la programación dinámica[2] de la forma más simple posible. Esta técnica se concibió para aprovechar los recursos que nos ofrece el entorno.

Dado un usuario que se quiere mover de una habitación a otra, la forma de detección de dicho movimiento se hace a través de un objeto del juego (`Cube`) que cubre por completo el lugar al que se quiere desplazar. Se puede ver en la Figura 5.2

Figura 5.2: Método por recubrimiento de superficie

Cuando el usuario se aproxima al cubo y colisiona con él, el *collider* asociado detecta la colisión y lanza un evento que lo trata el método `OnTriggerEnter()`. El evento consiste en un mensaje informativo que le indica al usuario el sitio donde se encuentra. Al salir fuera del alcance del radio de acción del cubo desparece el mensaje y con ello la información donde se encantraba el

[2]Intenta resolver problemas disminuyendo su coste computacional aumentando el coste espacial.

usuario.

Este método sería totalmente válido por su sencillez si no fuese por los siguientes problemas:

- *Mensaje persistente*: si el usuario se mantiene dentro del alcance del cubo aparecerá el mensaje continuamente hasta que salga de él. Por el contrario, si se desactiva dentro de la función de `OnTriggerOn()` el mensaje después de un determinado tiempo, no se mostraría más y perderíamos la ubicación.
- *Superficie irregular*: si hay un lugar que no tenga una superficie con forma de polígono regular, existe la posibilidad de que el cubo que recubre la habitación se pueda quedar corto o largo de tamaño. Esto provocaría que el usuario pudiese acceder a una parte de la estancia que no está dentro del recubrimiento del cubo y por tanto, perderíamos indefinidamente su posición. Por otra parte, si el usuario se encuentra en una sala contigua a la que tiene este problema, podría darse el caso de que el cubo sobresaliese de forma que detectase su presencia, informándole erróneamente de su ubicación. Con este método estaríamos obligados a poner cubos pequeños para simular la forma irregular de la zona.
- *Sobrecarga espacial*: este fenómeno está ligado al número de sitios que conforman la facultad que al ser uno tan elevado desborda el juego con una cifra desproporcionada de cubos.

5.4.2. Método por cálculo vectorial

El segundo método consiste en calcular mediante operaciones matemáticas si el usuario está dentro del cubo. La forma más sencilla es calcular la ecuación del plano para cada una de las seis caras que forman el cubo. Se trazan desde el punto las correspondientes normales a cada cara de este y se comprueba que el signo del resultado sea positivo (en el caso de que esté dentro) o negativo (si está fuera).

Este procedimiento se descartó, ya que había otros métodos que implementaban de forma interna estas operaciones tan complejas.

5.4.3. Método por duplicidad

Es una variante del primer método que consiste en situar un cubo de tamaño pequeño por delante y otro por detrás de la puerta (ver Figura 5.3). Estos objetos informan de la posición del usuario en la sala actual en la que se encuentra, en el momento en que se detecta su colisión con el cubo.

Este procedimiento conlleva los siguientes problemas:

- *Orientación*: solo cuando el jugador entra en colisión con alguno de los cubos, se conoce su posición. Es decir, la ubicación del jugador es desconocida mientras interactúa dentro de la habitación. Por otro lado, como se puede ver en la Figura 5.3, la separación entre ambos cubos es la mínima como para que no se superpongan los mensajes que se muestran al jugador durante el juego. En este espacio también se perdería la posición si decide quedarse en medio. El otro problema se produce al salir, ya que el jugador volverá a entrar en contacto con el `collider` mostrando de nuevo la situación en la que se encontraba.
- *Sobrecarga espacial*: es idéntico al que se produce en el *método del recubrimiento espacial* pero con el doble de cubos y de menor tamaño.

El primer problema se resuelve añadiendo una variable global que guarda el estado actual del jugador y que cambiará cuando el usuario colisione con algún cubo. En el caso de que la ubicación sea distinta se actualizará el valor de la variable con la nueva posición, en caso contrario conservará el mismo que tenía.

Este método es el más efectivo y el que presenta menos problemas. Por estos motivos es el que se ha implementado.

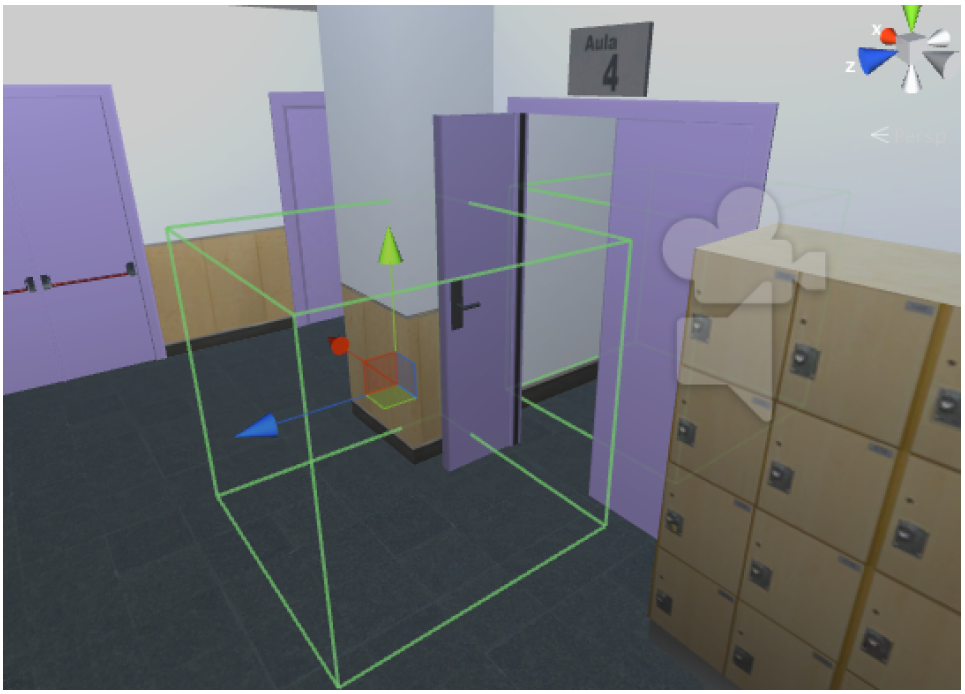


Figura 5.3: Método por duplicidad

5.5. Carga inicial de la aplicación

La necesidad de estructurar el juego en diferentes escenas planteó la pregunta de cómo se podía realizar la carga de los objetos necesarios. La generación de historias, personajes y mapeado de la facultad conlleva poder cargar de forma dinámica un conjunto de cubos en determinadas posiciones, los cuales indican su desarrollo.

En una primera aproximación, la implementación se realizó con objetos integrados de forma estática en el proyecto. Así se comprobó que la viabilidad del *método por duplicidad* era positiva. Para solucionar el problema de *sobrecarga espacial* se utilizaron ficheros externos que permitían la carga de los objetos necesarios de forma automática.

El principal formato utilizado para los archivos cargados externamente es XML dado que proporciona una jerarquía de elementos estructurada y comprensible. Al principio solo había un único archivo XML, el cual contenía tanto el mapeado de la facultad como el de los personajes que estaban dependientemente relacionados. Posteriormente, se concluyó que la mejor manera de recabar los datos era dividiendo la estructura para maximizar la independencia entre archivos. Dentro de la carpeta *Resources* del proyecto se encuentran los archivos XML utilizados, son ocho: `facultad.xml`, `historia.xml`, `objetivos.xml`, `personajes.xml`, `preguntas.xml`, `mensajesControl.xml`, `protagonistas.xml` y `objetos.xml`.

5.5.1. Facultad

Este archivo `facultad.xml` contiene la ubicación de los cubos que definen el mapeado de la facultad. Además consta de un contenedor que alberga todas las plantas dentro de las cuales se definen diferentes lugares de la facultad identificados unívocamente por un ID. Cada lugar tiene asociado un tipo, referente a si es un pasillo, escalera o estancia[3] y la orientación en la que está situado[4]. Las entradas constan de un identificador, del nombre del lugar al que pertenecen y un campo de adyacencia que indica el tipo de habitación del lugar al que dan acceso. También incluye el orden de la posición que ocupa el lugar respecto a otros lugares que tienen el mismo campo *tipo* en la misma planta[5] y una posición que indica donde se encuentra el objeto que representa dicha entrada. El archivo tiene la siguiente estructura:

[3] El tipo Estancia lo conforman todas las aulas, laboratorios, despachos, baños, etc. El cometido de la división de los tipos de habitación se explicará en el apartado 7.3.

[4] La orientación es relativa respecto al plano del edificio de la facultad de la Figura 7.1.

[5] Por ejemplo: en la planta baja, el orden determina que para el caso de las aulas, el aula 1 es la precedente al aula 2.

```
<Contenedor>
  <PlantaBaja>
    <Lugar id="cafe1" tipo="estancia" orientacion="E">
      <Entradas>
        <Entrada id="eNorte" nombre="cafeteria" adyacente="pasillo">
          <Orden>0</Orden>
          <Posicion>
            <X>2.557</X>
            <Y>0.934</Y>
            <Z>7.313</Z>
          </Posicion>
        </Entrada>
        <Entrada id="eSur" nombre="cafeteria" adyacente="pasillo">
          <Orden>0</Orden>
          <Posicion>
            <X>27.562</X>
            <Y>0.984</Y>
            <Z>6.257</Z>
          </Posicion>
        </Entrada>
      </Entradas>
    </Lugar>
  </PlantaBaja>
  <PrimeraPlanta>
    <Lugar id="escaleraN" tipo="escalera" orientacion="N">
      <Entradas>
        <Entrada id="ePPO" nombre="escalera de la primera planta norte"
          adyacente="pasillo">
          <Orden>0</Orden>
          <Posicion>
            <X>-6.208</X>
            <Y>3.816</Y>
            <Z>-0.509</Z>
          </Posicion>
        </Entrada>
      </Entradas>
    </Lugar>
  </PrimeraPlanta>
</Contenedor>
```

5.5.2. Historia

El archivo consta de un contenedor que alberga las diferentes historias que se cargan en el proyecto, además de la posición inicial de la que partirá el personaje independientemente de cómo se desarrollen las historias.

La posición inicial viene dada por el número de planta, el lugar y la puerta en la que aparecerá el personaje. Cada historia almacena todos los sectores de la facultad en los que se desarrolla, que son identificados con el mismo ID

que los lugares de `Facultad.xml`.[6] Cada sector contiene una lista de puntos u objetivos que el personaje debe cumplir para completar la historia. Los puntos de control son identificados por un ID y el orden en el que han de completarse para terminar cada historia. Además, contienen el mensaje que se le mostrará al usuario y que contará el narrador cuando el jugador pase por el punto anteriormente indicado.

El archivo tiene la siguiente estructura:

```
<ContenedorHistorias>
  <PosicionInicial planta="baja" lugar="cafe1" puerta="eSur"/>
  <Historias>
    <Historia id="Historia1">
      <Lugares>
        <Sector id="cafe1">
          <PuntosControl>
            <Punto id="Punto1" orden="1">
              <Mensaje>Va a ser la hora de comer.</Mensaje>
            </Punto>
            <Punto id="Punto2" orden="2">
              <Mensaje>¡Vaya! Son casi las dos.</Mensaje>
            </Punto>
          </PuntosControl>
        </Sector>
        <Sector id = "pO">
          <PuntosControl>
            <Punto id="Punto3" orden="3">
              <Mensaje>Tendrás que darte prisa para llegar a
                  clase.</Mensaje>
            </Punto>
          </PuntosControl>
        </Sector>
      </Lugares>
    </Historia>
  </Historias>
</ContenedorHistorias>
```

5.5.3. Objetivos

El archivo contiene los detalles de todos los objetivos posibles que se pueden incluir en una historia. Cada objetivo se identifica por un ID y contiene información relativa al `GameObject` que está asociado a dicho objetivo. Concretamente se almacena la dimensión y la posición donde se encontrará el objeto una vez cargada la historia dadas las coordenadas X, Y y Z que necesita la clase `Vector3` para poder instanciar el objeto en el juego.

El archivo tiene la siguiente estructura:

[6] Para hacer referencia a los elementos Lugar de `Facultad.xml` y no provocar errores de colisión de nombres durante el proceso de carga, se cambió el nombre a Sector.

```
<ContenedorObjetivos>
  <Objetivos>
    <Objetivo id="Punto1">
      <Dimension>
        <X>1.9</X> <Y>1.0</Y> <Z>0.6</Z>
      </Dimension>
      <Posicion>
        <X>26.518</X> <Y>0.92</Y> <Z>6.373</Z>
      </Posicion>
    </Objetivo>
    <Objetivo id="Punto2">
      <Dimension>
        <X>0.70</X> <Y>0.95</Y> <Z>1</Z>
      </Dimension>
      <Posicion>
        <X>16.5</X> <Y>0.92</Y> <Z>10.96</Z>
      </Posicion>
    </Objetivo>
  </Objetivos>
</ContenedorObjetivos>
```

5.5.4. Objetos

El archivo contiene los distintos objetos que se introducirán en el juego. Cada uno de ellos está constituido por un ID, la planta en la que estarán y el modelo. El archivo tiene la siguiente estrutura:

```
<ContenedorObjetos>
  <Objetos>
    <Objeto id="1" planta="0" modelo="Coca-Cola">
      <Posicion>
        <X>10.38</X> <Y>1.18</Y> <Z>11.47</Z>
      </Posicion>
      <Rotacion>
        <X>0</X> <Y>0</Y> <Z>0</Z>
      </Rotacion>
    </Objeto>
    <Objeto id="2" planta="0" modelo="Prefap_Monster">
      <Posicion>
        <X>23.166</X> <Y>0.65</Y> <Z>7.572</Z>
      </Posicion>
      <Rotacion>
        <X>0</X> <Y>0</Y> <Z>0</Z>
      </Rotacion>
    </Objeto>
  </Objetos>
</ContenedorObjetos>
```

El archivo consta de la posición y rotación que adquirirán al instanciarlos con el fin de que el usuario pueda colocarlos en la forma que desea.

5.5.5. Personajes

La estructura que conforma el archivo `personajes.xml` contiene tanto la ubicación como la información descriptiva de los personajes. El color de la ropa tanto superior como inferior y del calzado, e incluso la posición y rotación que va a adoptar el personaje dentro del juego son las características que vienen especificadas en el documento.

Conviene destacar que estos personajes son los de relleno, es decir, que no afectan al desarrollo del juego. El archivo tiene la siguiente estructura:

```
<ContenedorPersonajes>
  <PersonajesExtras>
    <PersonajeExtra id="pe1" planta="0" modelo="Prefap_ChicaSentada"
    animado="no">
      <Posicion>
        <X>25.406</X> <Y>−0.225</Y> <Z>−9.421</Z>
      </Posicion>
      <Rotacion>
        <x>0</x> <Y>0</Y> <Z>0</Z>
      </Rotacion>
      <Modelizar>
        <RopaSuperior>rojo</RopaSuperior>
        <RopaInferior>blanco</RopaInferior>
        <Calzado>negro</Calzado>
      </Modelizar>
    </PersonajeExtra>
    <PersonajeExtra id="pe2" planta="0" modelo="Prefap_ChicaSentada"
     animado="no">
      <Posicion>
        <X>20.406</X> <Y>−0.225</Y> <Z>−9.421</Z>
      </Posicion>
      <Rotacion>
        <x>0</x> <Y>8.188</Y> <Z>0</Z>
      </Rotacion>
      <Modelizar>
        <RopaSuperior>magenta</RopaSuperior>
        <RopaInferior>gris</RopaInferior>
        <Calzado>rojo</Calzado>
      </Modelizar>
    </PersonajeExtra>
  </PersonajesExtras>
</ContenedorPersonajes>
```

El archivo consta de un lista de personajes que incluye los identificadores, la planta en la que se encuentran, el `prefab` de cada uno y si están animados.

5.5.6. Protagonistas

La estructura que conforma el archivo `protagonistas.xml` contiene tanto la ubicación como la información descriptiva del protagonista que será el que forme parte de las historias. La posición y rotación que va a adoptar dentro del juego, el mensaje que va a decir cuando se acerque el jugador, los objetivos a los que se va a dirigir y el color de la ropa son las características que vienen específicadas en el documento. El archivo tiene la siguiente estructura:

```
<ContenedorProtagonistas>
  <PersonajesPrincipales>
    <PersonajePrincipal id="pp1" idHistoria="Historia1"
    idObjetivo="Punto1" nombre="Laura García" camara="si"
    modelo="Prefap_ChicaDePie1" animado="si">
      <Posicion>
        <X>18.537</X>
        <Y>0.11</Y>
        <Z>6.43</Z>
      </Posicion>
      <Rotacion>
        <X>0</X>
        <Y>0</Y>
        <Z>0</Z>
      </Rotacion>
      <Hablar>
        Necesitas dirigirte a los laboratorios, allí encontrarás
        lo que buscas.
      </Hablar>
      <ObjetivosSeguir>
        <Posicion id="1">
          <X>16.378</X>
          <Y>0.7</Y>
          <Z>6.43</Z>
        </Posicion>
        <Posicion id="2">
          <X>12.944</X>
          <Y>0.7</Y>
          <Z>10.397</Z>
        </Posicion>
      </ObjetivosSeguir>
      <ModelizarFigura>
        <RopaSuperior>
          <R>62</R>
          <G>95</G>
          <B>138</B>
        </RopaSuperior>
        <RopaInferior>
          <R>36</R>
          <G>231</G>
          <B>17</B>
        </RopaInferior>
        <Calzado>
          <R>0</R>
```

```
            <G>0</G>
            <B>0</B>
          </Calzado>
        </ModelizarFigura>
      </PersonajePrincipal>
      <PersonajePrincipal id="pp2" idHistoria="Historia1"
       idObjetivo="Punto2" nombre="Pilar Claro" camara="no"
       modelo="Prefab_ChicaDePie" animado="si">
        <Posicion>
          <X>20.374</X>
          <Y>0.11</Y>
          <Z>6.476</Z>
        </Posicion>
        <Rotacion>
          <X>0</X>
          <Y>87</Y>
          <Z>0</Z>
        </Rotacion>
        <Hablar>
          ¡Me has encontrado! Está bien, busca a Laura.
          Lleva una camisa rosa. Ella te dirá qué hacer.
        </Hablar>
        <ObjetivosSeguir>
        </ObjetivosSeguir>
        <ModelizarFigura>
          <RopaSuperior>
            <R>248</R>
            <G>243</G>
            <B>53</B>
          </RopaSuperior>
          <RopaInferior>
            <R>202</R>
            <G>196</G>
            <B>176</B>
          </RopaInferior>
          <Calzado>
            <R>197</R>
            <G>29</G>
            <B>52</B>
          </Calzado>
        </ModelizarFigura>
      </PersonajePrincipal>
    </PersonajesPrincipales>
</ContenedorProtagonistas>
```

El archivo está formado por una lista de protagonistas que incluyen un identificador por cada uno, el identificador de la historia a la que pertenecen, con su respectivo identificador del punto de la historia donde se instanciarán, el nombre, si va a ser seguido por una cámara cuando entre en escena, el `prefab` que se cargará y si es animado o no.

5.5.7. Mensajes de control del narrador

El archivo consta de los mensajes que el narrador utilizará para intentar hacer que el usuario se dirija al punto deseado, con distintos niveles de intensidad que dependerán de si el jugador decide rehusar las indicaciones dadas. Cada uno de estos niveles está definido por un identificador y por distintas frases, aperturas y cierres. La estructura del archivo es la siguiente:

```
<ContenedorFrases>
  <Niveles>
    <Nivel id="1">
      <Frases>
        <Frase sitio="no">
          <Mensaje>Gabi decidió ir en dirección opuesta</Mensaje>
        </Frase>
        <Frase sitio="no">
          <Mensaje>Gabi sintió curiosidad y tuvo la necesidad de
          desviarse</Mensaje>
        </Frase>
      </Frases>
      <Aperturas>
        <Apertura minus="no">
          <Mensaje>¡Vaya!</Mensaje>
        </Apertura>
      </Aperturas>
      <Cierres>
        <Cierre minus="si">
          <Mensaje>haciendo caso omiso a mis indicaciones</Mensaje>
        </Cierre>
      </Cierres>
    </Nivel>
    <Nivel id="2">
      <Frases>
        <Frase sitio="actual">
          <Mensaje>Gabi se dirigió</Mensaje>
        </Frase>
      </Frases>
      <Aperturas>
        <Apertura minus="coma">
          <Mensaje>En un alarde de rebeldía</Mensaje>
        </Apertura>
      </Aperturas>
      <Cierres>
        <Cierre minus="punto">
          <Mensaje>Dolida me siento por su comportamiento</Mensaje>
        </Cierre>
        </Cierre>
      </Cierres>
    </Nivel>
  </Niveles>
</ContenedorFrases>
```

5.5.8. Estructura de preguntas

La estructura que se presenta a continuación del archivo `preguntas.xml` incluye las preguntas de un test que se realiza en un minijuego. Contiene las preguntas con sus respectivos identificadores, enunciados y respuestas.

```
<ContenedorPreguntas>
  <Preguntas>
    <Pregunta id="p1">
      <Enunciado>¿Cuál es la capital de Bielorrusia?</Enunciado>
       <Opciones>
         <Opcion num="1" correcta="no">
          <Nombre>Sarajevo</Nombre>
         </Opcion>
         <Opcion num="2" correcta="si">
          <Nombre>Mínsk</Nombre>
         </Opcion>
         <Opcion num="3" correcta="no">
          <Nombre>San Petersburgo</Nombre>
         </Opcion>
         <Opcion num="4" correcta="no">
          <Nombre>Ulster </Nombre>
         </Opcion>
        </Opciones>
    </Pregunta>
    <Pregunta id="p2">
      <Enunciado>¿Donde se ubica el volcán Mauna Kea?</Enunciado>
       <Opciones>
         <Opcion num="1" correcta="no">
          <Nombre>Islas Malvinas</Nombre>
         </Opcion>
         <Opcion num="3" correcta="no">
          <Nombre>Japón</Nombre>
         </Opcion>
        </Opciones>
    </Pregunta>
  </Preguntas>
</ContenedorPreguntas>
```

Las respuestas están enumeradas y constan de un campo que indica si es correcta o no.

5.5.9. Propiedades comunes entre los archivos XML

Una de las características que tiene en común tanto `facultad.xml` como `personajes.xml` es la posición, la cual esta formada por las coordenadas X,Y y Z que se almacenan como un `Vector3`. Describe la posición que van a ocupar los `prefabs` de personajes y de los cubos.

En el caso de los personajes, los `prefabs` siguen un modelo humanoide que

distingue partes del cuerpo humano de forma independiente. Es decir, se pueden colocar las distintas extremidades en función de la rotación de estas. Con ello se consigue moldear a un personaje en distintas posturas. En el caso de los cubos se utilizan como área de control dentro del juego. Esto se consigue haciendo transparente el cubo desactivando el `Mesh Renderer` y añadiéndole el `script` de comportamiento.

5.6. Minijuego

Como diría Walter Scott: "*El que sube una escalera debe empezar por el primer peldaño*". Este mismo concepto es el que aplicamos utilizando un diseño *bottom up*. La realización de este planteamiento empieza por aprender algo sencillo para después poder realizar algo más complejo.

Por ello se ideó un primer minijuego llamado *Prueba de Ingenio* para interactuar con el usuario y aprovechar los recursos del modelo de la facultad. El juego pretende simular un cuestionario con preguntas de tipo test sobre cultura general básica modelado sobre la pizarra de un aula.

5.6.1. Desarrollo

La primera idea que se intentó implementar consistía en utilizar en la escena objetos `3D Text` con componentes `Text Mesh` para visualizar tanto las preguntas como las respuestas del minijuego. El problema que se observó fue que el texto atravesaba cualquier estructura del edificio como si fuera transparente, por lo que todos los componentes del cuestionario, tanto preguntas como respuestas, eran visibles desde cualquier extremo de la facultad. Además, implementar la funcionalidad para que el usuario pudiera interactuar era excesivamente complicada debido a que no proporciona elementos que permitan activar eventos, como botones.

Debido a este inconveniente optamos por utilizar el sistema `Canvas` que agrupa elementos `UI` de forma jerárquica, gracias al cual pudimos gestionar los siguientes componentes utilizados en la Figura 5.4:

- *Text*: se usa para establecer los enunciados de las preguntas y sus respectivas respuestas.
- *Button*: se utiliza para saber la respuesta que ha marcado el usuario, dentro de cada botón se encuentra un componente `Text` que define su contenido.
- *Panel*: se emplea para organizar a los componentes de forma que se conserve una estructura jerárquica.

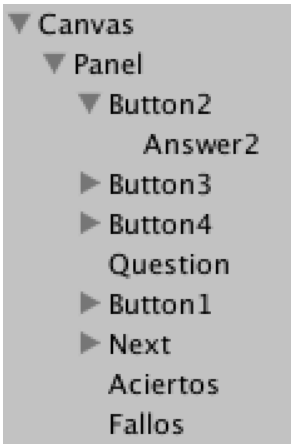


Figura 5.4: Modelo Canvas en Unity

En una primera aproximación, se intentó generar de forma dinámica, es decir, desarrollar el modelo y los componentes anteriores desde código para construir el `Canvas` que representa la interfaz de las preguntas. El procedimiento se llevó a cabo con éxito, salvo que la distribución resultante del proceso no es la que se pretendía seguir. El texto de las preguntas y de las respuestas no adoptaban la posición deseada en la pizarra por lo que se decidió implementarlo de forma estática.

El resultado final, como se puede ver en la Figura 5.5, es una estructura que consta de tres componentes `Text`, uno que muestra el número de aciertos, otro el número de fallos y otro la pregunta en sí. También dispone de dos tipos de botones, uno que contiene las posibles respuestas que el jugador puede seleccionar y otro para verificar la respuesta y continuar cuando se escoja, pero hasta que no sea marcada el botón permanecerá desactivado.

Figura 5.5: Escena Canvas en Unity

Las preguntas de dicho test se cargan desde un archivo `XML` donde están definidas. Cada una de ellas está formada por cuatro respuestas de las cuales sólo una es correcta. De este modo para cambiar a otra pregunta se reutiliza el modelo modificando las componentes con el texto de los enunciados y respuestas.

Desde que se empezó a trabajar en el juego se quiso combinar elementos 3D con elementos 2D dando así un aspecto arriesgado y creativo, aunque eso supone dar una mayor complejidad a su desarrollo. Una de las partes en las que se pretendía profundizar es en la perspectiva del jugador. La visión que tiene es un plano frontal de la pizarra como se puede ver en la Figura 5.6, de tal forma que da la apariencia de estar en dos dimensiones.

Para lograrlo se creó otra escena, la cual nos permitió situar la cámara fija e impedir que el personaje se pudiera mover para mostrarle el plano deseado.

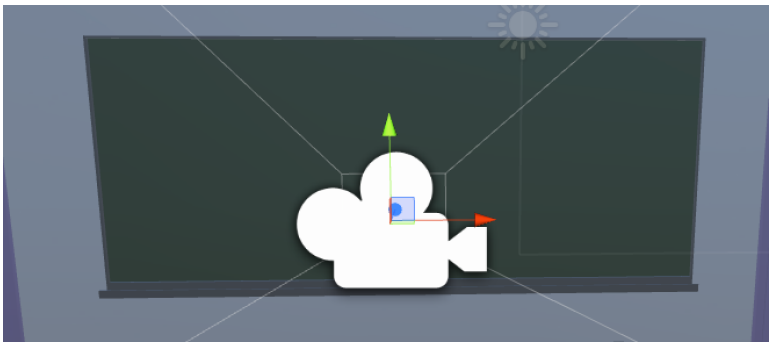
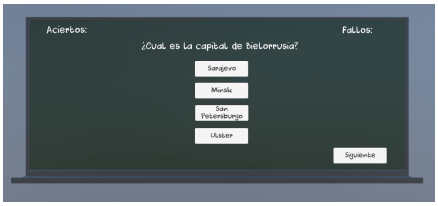


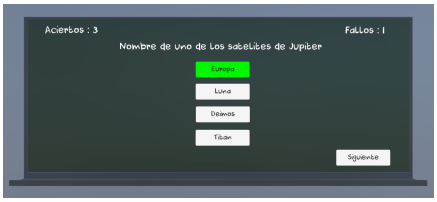
Figura 5.6: Perspectiva pizarra en Unity

Tal como muestra la imagen 5.7, si la respuesta es correcta se resaltará el botón que la contiene con el color verde. Si por el contrario la respuesta es errónea, el color seleccionado será el rojo. Al finalizar el test se calcula el porcentaje de aciertos que ha obtenido y será mostrado junto con un comentario que dependerá del resultado:

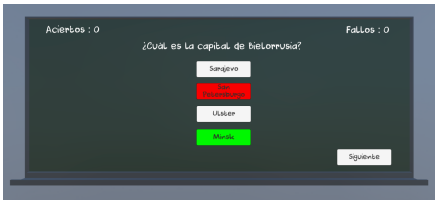
- Si ha contestado erróneamente a todas las preguntas se le mostrará el mensaje *"Vuelve al colegio"*.
- En el caso de que haya contestado correctamente a menos de la mitad de las preguntas, pero al menos una bien, se le mostrará el siguiente mensaje *"Necesitas estudiar más, cómprate un libro"*.
- Suponiendo que el jugador conteste más del 50 % correctamente sin llegar al 100 % se le mostrará *"No está mal, sigue así"*.
- Si responde a todas las preguntas adecuadamente se le mostrará *"¡Eres un genio y lo sabes!"*.



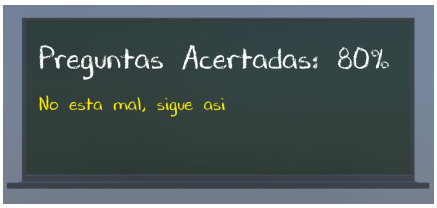
(a) Preguntas Iniciales



(b) Pregunta Acertada



(c) Pregunta Fallada



(d) Resultado

Figura 5.7: Ejemplo del minijuego

Capítulo 6

Desarrollo de las historias

Si buscas resultados distintos no hagas siempre lo mismo.
Albert Einstein

6.1. Introducción

Tras realizar los cimientos del proyecto constituimos una base que nos permitió realizar algo más grande. En este capítulo trataremos aspectos de gran envergadura que nos han llevado a plantearnos distintos caminos para lograr la generación de las historias.

En este apartado en particular nos centraremos en los métodos que hemos tomado para realizar las historias, desde la idea original para desarrollar una mediante cubos que explicaremos más adelante, junto con los problemas que fueron surgiendo. Se verá también la explicación de cómo se pueden introducir mensajes en la historia dependiendo de la distancia o el tiempo.

Hablaremos de cómo tuvimos que reestructurar la mayor parte del proyecto para poder integrar el desarrollo de historias y las relaciones que surgen entre ellas. Además, este nuevo cambio debe ser compatible con el *Método de duplicidad* explicado en el apartado 5.4 para relacionar ambos sistemas y sacar un mayor rendimiento a la aplicación. Las estructuras que se tuvieron en cuenta a la hora de implementar estas historias presentan ventajas, inconvenientes y partes comunes que también trataremos.

6.2. Primeros pasos de la historia

Al principio se pensó como se podía crear una historia sencilla que tuviera lugar dentro de la facultad mediante el uso de algún objeto o método que permitiese narrar el guión de la historia desde un archivo. Lo más lógico era utilizar los objetos `Cube` que ya estaban implementados para detectar la posición en la que se encontraba el jugador. Por eso, en un primer lugar se aumentó el número de dichos cubos, aunque esto supuso un gran incremento del coste en memoria. En este caso, dichos cubos se dotaron de un mensaje para que se visualizase por pantalla indicando al jugador el siguiente objetivo a completar o la información pertinente de la historia.

Para poder mostrar estos mensajes y diferenciarlos de los que se mostraban cuando el jugador cambiaba de lugar nos vimos en la necesidad de crear un nuevo estilo, y pensar cómo queríamos que se mostrase por pantalla. Elegimos situarlo en la parte superior izquierda como se ve en la Figura 6.1, análogamente a algunos videojuegos que lo sitúan en ese mismo lugar, siendo visible pero no molesto. Para este estilo ya no se emplea el sistema `Canvas` utilizado para otras partes del proyecto como en el *Minijuego 5.6* , sino que ahora se hace uso de los componentes `GUI` que nos ofrece *Unity* y el método `OnGUI()` que se encarga de gestionarlos en tiempo real. Aunque se consideró la posibilidad de usar el método `Canvas` no era necesario un estilo tan complejo e interactivo sino algo más sencillo e intuitivo.

Figura 6.1: Estilo GUI

Los mensajes de las historias se mostraban en función de la distancia que existía entre el jugador y el cubo al que se aproxima que contenía dicho mensaje. Para ello se utilizó `Vector3.Distance()` que nos devolvía el resultado

deseado, de tal forma que teníamos una restricción que indicaba que, si el jugador se situaba a una distancia de unos tres metros, lo mostrase, y en el caso de que fuese mayor desapareciese. Se escogió una distancia mayor que las dimensiones de los cubos para que el personaje no tuviera que atravesarlo completamente y así activase el mensaje. De esta forma no era necesario cambiar el tamaño de los cubos en función de la superficie que se pretendía abarcar, estableciendo así un `Prefab` que cumple el mismo cometido para todas las historias.

El resultado que conseguimos fue el esperado, pero no se habían contemplado muchos aspectos que afectaban en el transcurso de la historia. El mayor inconveniente era que no se podía distinguir entre los mensajes pertenecientes a la historia y los que advertían de la ubicación del personaje. En otras palabras, se desconocía si el jugador entraba en contacto con un cubo de historia o uno de posición. La consecuencia de ello repercutía en que no podíamos detectar los objetivos de la historia que el jugador había superado. Aunque a través del *Método por duplicidad* podíamos saber el lugar en el que estaba el jugador, no conocíamos el sitio al que se tenía que dirigir para continuar con la historia.

Para resolver el problema tuvimos que dividir en dos los tipos de cubos: *cubo de posición* y *cubo de historia*. Por un lado los cubos de posición, que constituyen la parte fundamental del mecanismo que permite conocer la ubicación del jugador en todo momento, y por otra parte los cubos de historia, que permiten narrar el desarrollo de la historia. Gracias a esta división podemos tratar los cubos por separado permitiéndonos mayor control y organización del código.

El mecanismo implementado para poder detectar el tipo de mensaje que tiene que mostrar se realiza mediante el `script Mensaje.cs` asociado al cubo con el que ha colisionado el jugador. Se realiza de modo parecido a como actúa una interrupción con un procesador. Cuando se produce una interrupción, el procesador genera una suspensión temporal del proceso que se está ejecutando y empieza a ejecutar el código específico que contiene la interrupción. En nuestro caso, la función `Update()` de cada `script` que implementa `MonoBehaviour` es el proceso que se está ejecutando continuamente. Cuando el personaje entra en contacto con un cubo de historia o con uno de posición activa el método `OnTriggerEnter()` que se ejecuta indefinidamente mientras el personaje se encuentra dentro de la superficie del cubo. La *interrupción* se produce cuando se detecta que el objeto con el que ha colisionado el personaje es algún cubo de historia o posición. Esta comprobación se realiza desde el `script CargarFacultad.cs` cuya función principal es cargar todos los cubos de posición e historia que se registran en los archivos `Facultad.xml` y `Historia.xml` respectivamente. Cuando se verifica que se trata alguno de estos tipos de cubo, se accede al `script Mensaje.cs` para ejecutar otra sec-

ción de código distinta a la que se estaba ejecutando hasta entonces en la función `Update()` de dicho `script`. En el caso de que el objeto con el que haya colisionado resulte ser un cubo de historia, este nuevo trozo de código remitirá la nueva configuración de los mensajes a la función `OnGUI()` con un estilo diferente a la que tienen los cubos de posición. La función `OnGUI()` también ejecuta un bloque de código distinto que permite crear los componentes `GUI` necesarios para adoptar el estilo establecido para cada tipo de mensaje.

6.3. Tiempo y orden de aparición de mensajes

El tiempo es la cosa más valiosa que una persona puede gastar.

Theophrastus

Cuando el mecanismo del apartado 6.2 estaba preparado, se añadieron mejoras en la frecuencia de aparición del mensaje. La historia se desarrollaba correctamente a la vez que se informaba con precisión de la posición en la que se encontraba el personaje. Hasta entonces sólo se tenía en cuenta la distancia de los cubos a la que el jugador se encontraba. Esto provocaba que pudieran darse varias situaciones en las que el jugador interfiriera en el correcto funcionamiento del mecanismo. Por ejemplo, si el jugador decidiera permanecer parcial o totalmente dentro de un cubo de historia durante el tiempo que quisiera, el mensaje que se le mostraría permanecería activo todo el tiempo que estuviera dentro de él. También podría darse la situación de que atravesara demasiado deprisa el cubo provocando que el mensaje saliera durante un breve periodo de tiempo directamente proporcional a la velocidad con la que lo atravesara.

Este problema se solucionó mediante la integración de un contador que se activaba cuando el personaje colisionaba con un objeto de tipo cubo de historia y se decrementaba tras su activación hasta llegar a cero. La reducción del contador se hacía efectiva con la ejecución completa de cada *frame* en el que se ejecutaba la función `Update()` del `script Mensaje.cs` asociado al cubo. Como dicho `script` estaba integrado en todos los cubos del juego, todos los contadores tomaban el mismo valor al inicializarse. El temporizador tomó un valor inicial aproximado de unos 10 segundos permitiendo leer los textos más largos.

Ahora el jugador podía quedar dentro de un cubo y aunque estuviese a una distancia adecuada una vez que el tiempo expiraba, el mensaje desaparecía. Pero también surgieron otros problemas. Por ejemplo, si el jugador decidía ir a una velocidad rápida y pasaba por dos cubos de manera casi consecutiva se producía la situación de que el primer mensaje no había terminado cuando

se veía solapado por el mensaje del otro cubo. Otro contratiempo radicó en aquellos mensajes que eran muy cortos los cuales se mostraban en un tiempo superior al necesario. Esto supuso tomar la decisión de mostrar el mensaje el tiempo suficiente en relación a su tamaño, lo cual llevó a plantearnos buscar una forma de almacenar los mensajes para poder mostrarlos en orden.

6.4. Plantillas de historias

Solucionado el problema de la persistencia temporal del mensaje y su distancia de aparición, nos planteamos la siguiente pregunta con el problema del solapamiento del texto aún en mente: ¿Cómo podemos establecer las relaciones que surgen entre las historias? Abordamos tres posibilidades:

- `Mono-historia` : conformada por una única historia.
- `Árbol de historias` : conjunto de historias con un principio común que pueden ramificarse en otras y terminar con uno o varios finales.
- `Árbol invertido de historias`: conjunto de historias con uno o varios principios que pueden ramificarse y converger en uno o varios finales.

Dadas estas tres opciones tan diferentes a nivel de coste y programación a la hora de implementarlas, tuvimos que evaluar las ventajas e inconvenientes de cada una de ellas.

6.4.1. Mono-historia

En este caso se puede obviar la gestión de las relaciones entre las historias al ser inexistente puesto que sólo se tiene en cuenta una única historia principal. Es la opción más simple posible y la de menor coste.

Para llevarla a cabo no es necesario una estructura de historias (como se puede ver en la Figura 6.2) tan compleja como la del archivo `Historia.xml`. De hecho, al implementar una historia, todo el desarrollo del juego puede estar perfectamente controlado y saber en todo momento lo que puede hacer o no el protagonista. Para evitar la monotonía del juego y dar la sensación al usuario de tener un poco más de control, se han considerado introducir pequeñas ramificaciones que pueden surgir de la historia principal. No tienen una duración muy extensa. Si el jugador decide desviarse y tomar un camino alternativo que está disponible desde el punto de la historia en la que se encuentre, entonces la historia terminará cuando el jugador complete el objetivo.

Esta opción se descartó por la simplicidad y poco dinamismo que podía aportar al proyecto. Nuestro objetivo difería de algo simple, buscábamos realizar un sistema de historias complejo para poder integrar así una mayor interacción con el usuario y poder crear nuevas historias.

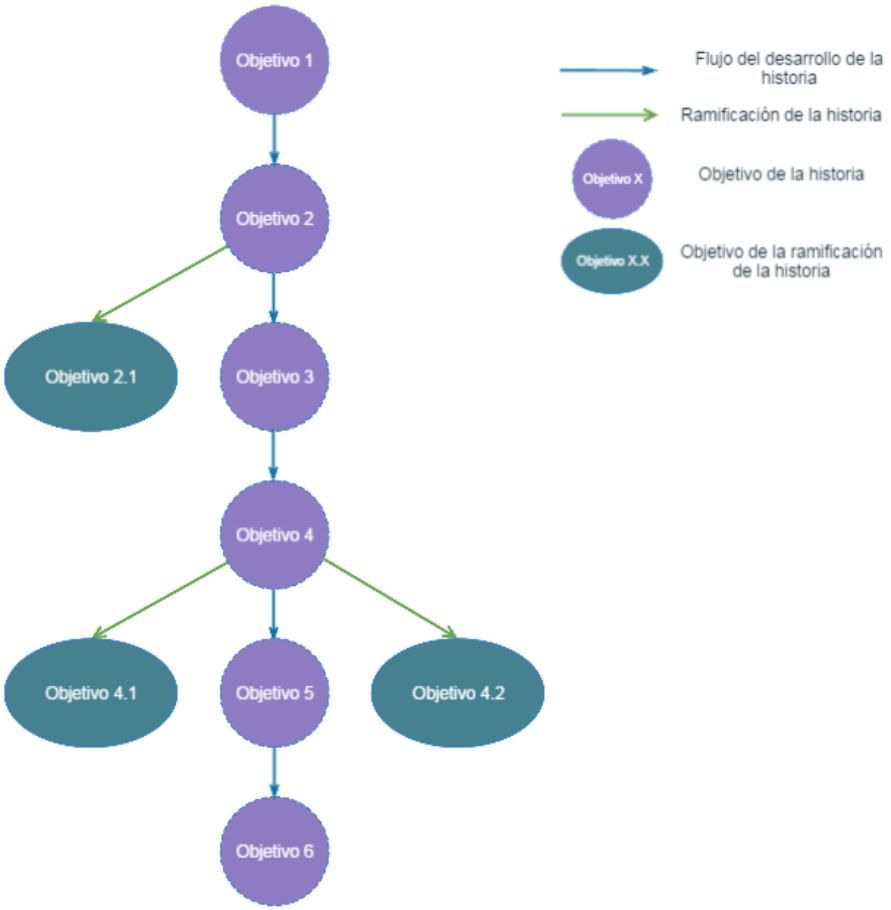


Figura 6.2: Estructura Mono-Historia

6.4.2. Árbol de historias

Es una estructura con forma de árbol (como se puede ver en la Figura 6.3) que empieza con un nodo inicial objetivo y que después se ramifica en otros dando lugar a otras historias. La relación entre ellas se establece al principio, teniendo todas el mismo punto de partida. Mediante esta organización podemos terminar de muchas formas posibles con diversos finales (como se ve en la Figura 6.4) e incluso se puede modificar la estructura para que todas converjan en uno (como sucede en la Figura 6.5).

A nivel de complejidad es mayor que la anterior, teniendo que hacer la carga dinámica de cubos para poder inicializar las nuevas historias. Para ello, se modifica la estructura de las historias del archivo `Historia.xml` para poder añadir los cubos que conformarán las nuevas historias. Cuando el jugador cambia de una historia a otra aparecerán uno o varios mensajes que servirán como nexo y con los que se relacionarán ambas historias para integrar al jugador de forma paulatina y comprensible.

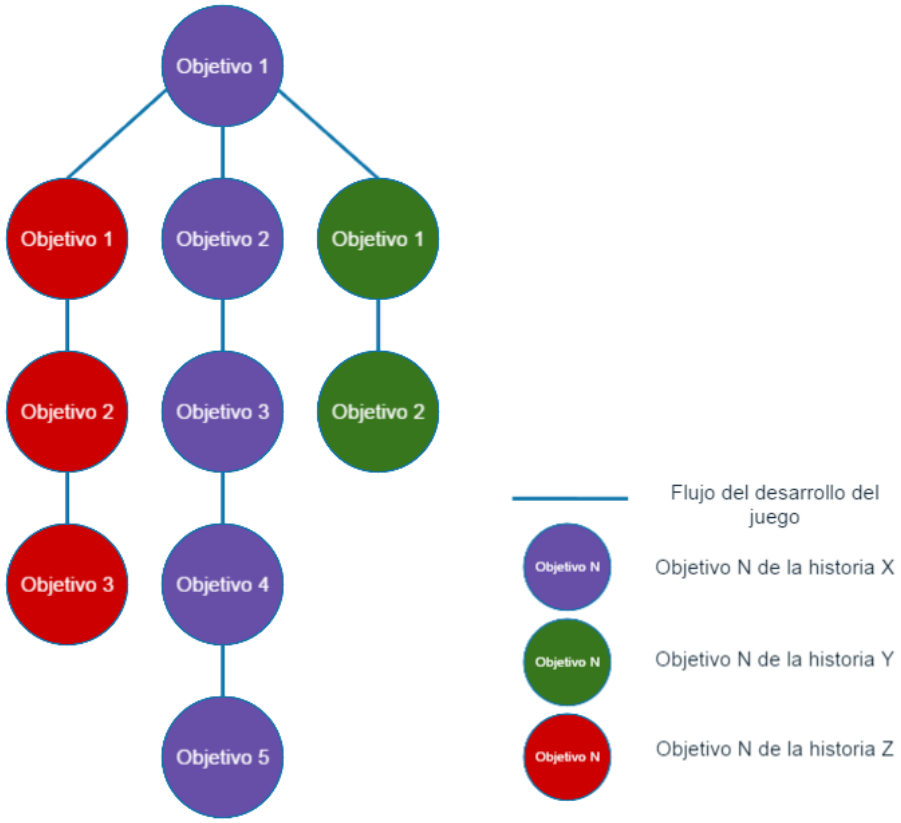


Figura 6.3: Árbol

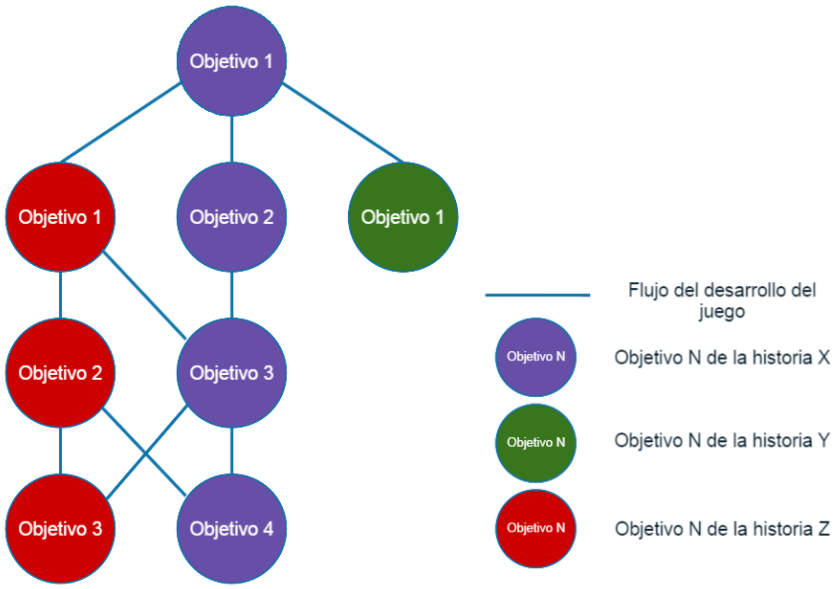


Figura 6.4: Árbol ramificado con diversos finales

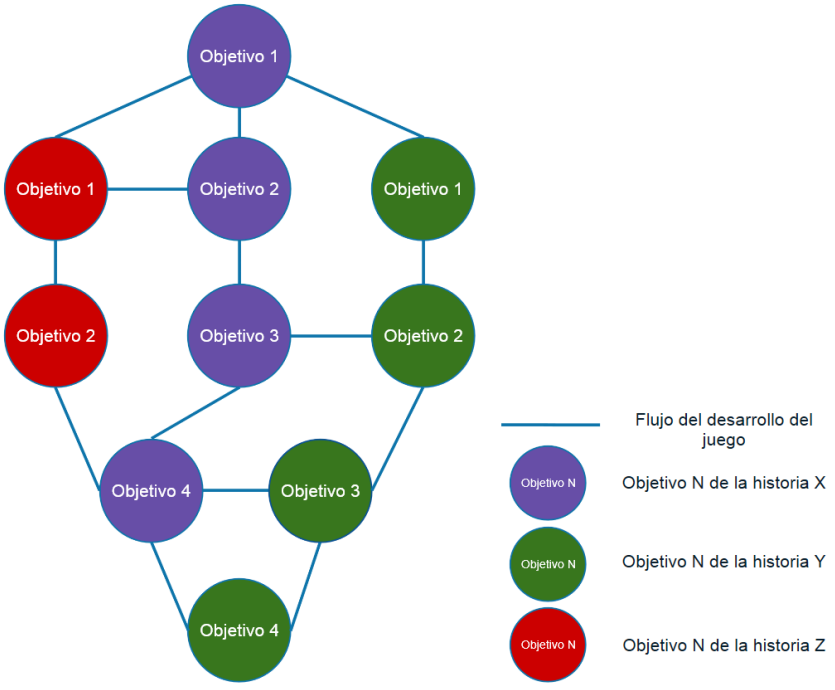


Figura 6.5: Árbol con ramificaciones que converge en un final

6.4.3. Árbol invertido de historias

La organización de este esquema es similar al que sigue el esqueleto del *Árbol de historias*. La principal diferencia consiste en la disposición inicial de los nodos (como se puede ver en la Figura 6.6), que permite que el jugador pueda escoger la historia por la que quiere empezar. El número de nodos objetivos iniciales es directamente proporcional al número de historias existentes en el archivo de `Historia.xml`. Es decir, si hay cinco historias definidas en dicho archivo, al empezar el juego se cargarán cinco posibles *cubos de historia* que el usuario podrá elegir para emprender su aventura gráfica narrativa.

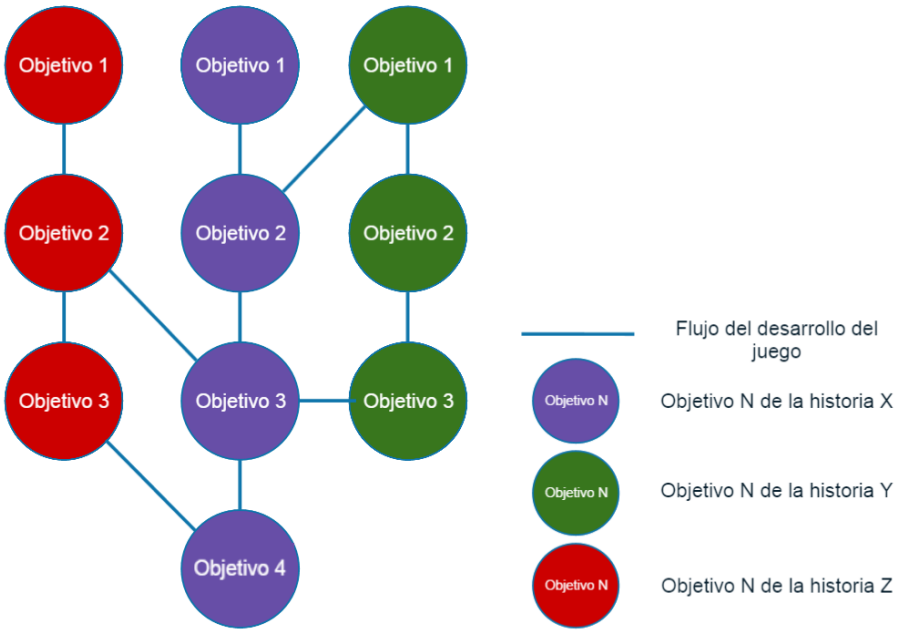


Figura 6.6: Árbol Invertido

Escogimos esta estructura, porque a diferencia del *Árbol de historias*, nos brinda la posibilidad de disponer de un abanico de historias al inicializarse el juego y no obligar al jugador a escoger una por defecto.

Imaginemos una situación en la que el usuario empieza a jugar y aparece en un punto dentro de la cafetería de la facultad. En el caso del *Árbol de historias*, nada más comenzar el jugador tendría asignado el primer objetivo de una historia que se ha estipulado como la inicial. Aunque el jugador intente desviarse hacia otros objetivos que no pertenecen a la misma, ya se le habrá introducido obligatoriamente en el nodo inicial. En cambio, con el *Árbol invertido*, cada vez que inicia el juego y aparece en la cafetería, no tiene

por qué empezar por el mismo nodo, sino que dependiendo del recorrido que elija hacer por la facultad podrá iniciar el desarrollo de diferentes historias que pueden comenzar en una determinada aula o incluso en las escaleras.

Otra de las razones para escoger este modelo es que permite llevar a cabo la implementación tanto de Mono-historia como la de Árbol de historias. En la estructura de tipo *Mono-Historia* las ramificaciones alternativas se corresponden en este sistema con las desviaciones hacia otras historias. Además, desde el punto de vista de las relaciones entre las historias, una estructura de tipo *Árbol Inverso* está formada por un *Árbol de historias* con la diferencia de que el nodo inicial no es común para todas ellas.

6.4.4. Propiedades comunes

Las decisiones que toma el jugador en el desarrollo de su historia pueden ser dos posibles opciones que dan lugar dependiendo de los modelos anteriores a diferentes bifurcaciones como veremos más adelante.

La primera opción se trata del caso ideal en el que el usuario interpreta las indicaciones que el narrador le va contando en los mensajes de historia. En este caso no se llegará a producir ningún evento imprevisto porque se entiende que el jugador completa la historia correctamente. La segunda se trata del caso alternativo en el que el jugador decide ignorar los consejos y sugerencias del narrador. Por ejemplo, el jugador puede optar por recorrer la facultad a modo de exploración. En este caso la historia no continuará hasta que el jugador la retome por el punto en el que la había dejado.

- `Mono-historia`:
  - Caso óptimo: cuando el personaje llega al final de la historia, está termina sin dar posibilidad de seguir el juego.
  - Caso alternativo: si el jugador decide desviarse y tomar un camino alternativo que está disponible desde el punto de la historia en la que se encuentre no podrá derivar en más, entonces finalizará el juego cuando el jugador complete el objetivo.
- `Árbol de historias y Árbol inverso de historias`:
  - Caso óptimo: si ha seguido la narración del juego sin desviarse se deduce que ha completado una historia de modo satisfactorio. Ahora puede decidir si quiere continuar ya que existen otras historias que se pueden desarrollar.
  - Caso alternativo: si el usuario toma una ramificación se le abrirá una gama de nuevas historias disponibles, las cuales se podrán

concatenar y retomar en cualquier momento formando una red de historias interconectadas.

La simplicidad del esquema *Mono-Historia* permite que los cubos de historia estén almacenados en memoria. Aunque estos puedan ser demasiados sólo pertenecen a una, por lo que suponen un coste relativamente bajo.

Respecto a las otras estructuras esta carga puede suponer un coste realmente alto porque no se sabe el alcance de ramificación de los nodos objetivos que pueden tener. Un ejemplo sería el caso de tener definidas un número de historias elevado cuyos cubos ocupan toda la facultad, suponiendo una sobrecarga innecesaria del sistema. Asimismo, al estar todos mapeados queda implícito que las historias se pueden entremezclar incorrectamente.

Además el jugador podría pasar de una a otra, únicamente si atraviesa cualquier cubo de la nueva e incluso podría repetir un objetivo. Es por ello que una vez el narrador ha terminado de informar al usuario del objetivo que tiene que seguir, el cubo que contiene dicho mensaje se elimina del juego para que no se pueda dar la situación previamente mencionada. De esta manera se evita la repetición de nuevos objetivos que ya han sido completados.

Implica un gasto inútil de recursos cargar todos los objetivos. Por ello, este inconveniente se solucionó siguiendo la siguiente premisa: el jugador debe poder acceder a todas las historias sin importar la posición en la que empiece. Si resulta ser en la planta baja debe ser capaz de comenzar otra historia que se encuentra en la última planta.

La solución que se llevó a cabo siguiendo la estructura *Árbol invertido* empieza instanciando todos los primeros cubos de todas las historias para que el usuario pueda tener varias formas de empezar como se explicó anteriormente. Durante el desarrollo de una se van cargando los cubos de uno en uno conforme el jugador va cumpliendo los objetivos. De esta forma sólo se carga el siguiente paso de la historia. A este método lo hemos llamado *Instancia dinámica*.

El coste anterior sin este método es $\sum_{i=1}^{n} N_i$, siendo $\mathrm{N_i}$ el número de cubos de la historia i y n el número de historias disponibles. En cambio con la *Instancia dinámica* se reduce a n.

Capítulo 7

Algoritmo de búsqueda de recorrido para detectar errores

El camino del éxito es el camino de la búsqueda continua del conocimiento.
Napoleon Hill

7.1. Introducción

En este capítulo se tratará la parte más compleja y más técnica del proyecto. Conseguido el desarrollo de una historia y elegida la estructura *Árbol Invertido*, había que dar el siguiente paso. Habíamos considerado la posición en la que se encontraba el jugador en todo momento y los siguientes nodos objetivo por los que tenía que pasar para completar la historia, pero no la ruta que tenía que tomar para llegar hasta dichos objetivos. O dicho de otro modo, desconocíamos si el jugador se desviaba del camino adecuado hasta el siguiente objetivo. Este concepto se ilustrará mediante ejemplos explicando los algoritmos utilizados para conseguir detectar el cambio de camino tomado por el usuario.

7.2. Evaluación del camino mínimo

La elección de la ruta hasta el siguiente objetivo debe ser en cualquier caso la mínima posible. Si no fuese así y pudiese escoger otro camino que no sea el mínimo, significa que el usuario podría tomar infinitas alternativas. Así para controlar los desvíos que puede tomar el jugador, basta con comprobar si la ruta por la que va se corresponde con la que se ha calculado como la menor

posible hasta el siguiente objetivo. Para realizar este cálculo se tuvieron en cuenta cuatro métodos:

- `Método de expansión`
- `Algoritmo de Bellman-Ford`
- `Algoritmo de búsqueda A* (A estrella)`
- `Algoritmo de Dijkstra`

7.2.1. Método de expansión

Este procedimiento es el primero que se consideró siendo el más sencillo. Consiste en aumentar el número de cubos de historia, situando algunos en posiciones estratégicas que son excluyentes de la historia principal. Es decir, no forman parte del transcurso de ésta sino que serán los que nos ayuden a controlar al jugador si decide cambiar de ruta. Los llamamos *cubos delimitadores*.

El inconveniente viene dado por la cantidad de cubos extra que sería necesario introducir para abarcar todas las posibles combinaciones de caminos que el usuario puede elegir, resultando no factible con el método de *Carga dinámica*, que permite solamente la carga de un cubo cada vez que el jugador pasa por un cubo de historia. Habría que cambiarlo o modificarlo para que los cubos de las historias siguieran este patrón, mientras que todos los delimitadores se tendrían que cargar desde el comienzo del juego junto con los *cubos de posición*.

Además se deben exigir restricciones respecto a la ubicación de los *cubos de historia*. Dependiendo de la cantidad de *cubos delimitadores* se puede conocer más o menos los posibles caminos alternativos que puede tomar el jugador pero persiste el problema de desconocer la ruta mínima hasta el objetivo. Por ello, los *cubos de historia* han de estar lo más cerca posible unos de otros, concretamente se debe exigir a lo sumo que se encuentren en la habitación contigua. De esta manera, el siguiente objetivo sólo es accesible desde una salida y por ello se sabe si el jugador decide desviarse de la ruta mínima.

Este método fue descartado por los problemas citados anteriormente, pero fundamentalmente porque es necesario conocer muchos detalles de la posición de los elementos impidiendo realizar una estructura para que otras personas puedan crear nuevas historias, que es al fin y al cabo uno de nuestros propósitos.

7.2.2. Algoritmos

Para realizar un sistema que permita encontrar el recorrido mínimo de una forma óptima, sin tener la necesidad de aplicar tantas restricciones de posición en los cubos de historia como sucede en el método anterior, se tuvo que usar los algoritmos de búsqueda del camino más corto explicados en el capítulo 4.3. Los nodos que conforman estos grafos son los cubos de posición por los que el jugador pasa y las aristas, las posibles rutas que puede tomar hacia otro lugar de la facultad.

El algoritmo de *Bellman-Ford* permite obtener este resultado pero cuenta con una serie de desventajas que los otros algoritmos no tienen. Una de las principales razones por la que no se eligió fue la *cuenta hasta el infinito*, es decir, que si se produce un fallo de enlace entre los *cubos de historia* generando que uno de ellos sea inalcanzable desde un conjunto de otros nodos objetivo, estos pueden estar siempre aumentando gradualmente sus cálculos de distancia respecto a él produciendo bucles infinitos.

Una mejor alternativa al algoritmo de *Bellman-Ford*, es el algoritmo de *Dijkstra* puesto que es capaz de resolver el problema de la búsqueda de la ruta más corta en un tiempo menor. Aunque es cierto que *Dijkstra* no es compatible con grafos de pesos negativos y *Bellman-Ford* sí, en el proyecto no se contempla esta implementación de costes ya que se obtienen por el valor absoluto de la distancia entre los cubos de posición que se verá más adelante. Por tanto, esta opción es la que más se adecua al proyecto con un coste máximo relativamente razonable siendo de $O(n^2)$ operaciones, donde n es el número de cubos de posición cargados en el juego.

También se consideró utilizar el algoritmo $A*$ cuya implementación y coste es muy parecido a *Djikstra*. No se escogió porque el número de caminos que debe de buscar no es demasiado grande y por tanto la velocidad de cálculo resulta casi despreciable, adoptando finalmente el algoritmo de *Dijkstra*.

7.3. Implementación del algoritmo de Dijkstra

El planteamiento seguido para llevar a cabo el procedimiento de *Dijkstra* requiere de una estructura que indique de alguna manera la relación que existe entre los cubos de posición. Al inicio se pensó en una función que determinaba esta concordancia mediante el análisis de la orientación que estos presentaban. En otras palabras, se comparaban los cubos teniendo en cuenta si estaban en la misma planta y en el mismo lugar que el resto. Para poder detectar la posición dentro de la planta se necesitó modificar el archivo que cargaba los cubos de posición llamado `CargarFacultad.xml`,

introduciendo la orientación que coincide con los puntos cardinales: Norte, Sur, Este y Oeste.

De esta forma, todos los cubos de un pasillo con orientación Este estarían relacionados, obteniendo la distancia que había entre cada uno de ellos y aparte estarían vinculados con los de la escalera norte o sur correspondiente. Llegando a ser una función que cumplía con su objetivo, nos dimos cuenta de que sólo servía para esta facultad, ya que, por ejemplo, únicamente valorábamos que había dos pasillos separados por una escalera cuando quizá otra facultad podía tener tres o más ligados de otra manera.

7.3.1. Matriz de adyacencia

Para realizar esta generalización y adaptar la funcionalidad a otros edificios, se decidió implementar una matriz que relacionese los cubos de posición al inicio del proyecto para después utilizar el algoritmo de *Dijkstra*. Esta matriz la definimos como la *Matriz de adyacencia*.

Las relaciones no deben estar tan sujetas al modelo de la facultad como sucede en el método inicial, sino que se requiere de muchas menos restricciones en cuanto a la posición de los cubos. Además, determinar esta relación basándose principalmente en la distancia que existe entre los cubos resulta ser poco precisa. Por estas razones, se quiere construir una matriz que permita conocer las relaciones que existen entre cubos de posición vecinos.

La matriz de adyacencia está sujeta a nuevas restricciones respecto al lugar donde se encuentran los cubos de posición vecinos. Los lugares a los que puede ir el personaje se reducen a tres grandes grupos: escalera, estancia y pasillo. La estancia conforma una zona que está formada por cuatro paredes, siendo homóloga a un cubo, como por ejemplo un despacho.

La implementación realizada consta de todos los casos posibles que se pueden dar, es decir, por cada escalera, estancia y pasillo se estudian los posibles vecinos que pueden tener. Siguiendo esta posible relación de vecindad:

- Origen de tipo *Escalera*:
  - Destino de tipo *Estancia*: una estancia no puede ser accedida directamente desde cualquier escalera.
  - Destino de tipo *Pasillo*: un pasillo puede ser accedido directamente desde una escalera.
  - Destino de tipo *Escalera*: una escalera no puede ser accedida directamente desde otra porque necesariamente debe pasar antes por un pasillo que dé a ella.

- Origen de tipo *Pasillo*:
  - Destino de tipo *Estancia*: una estancia se puede acceder directamente desde un pasillo.
  - Destino de tipo *Pasillo*: un pasillo no se puede acceder directamente desde otro porque necesariamente tendrían que tratarse del mismo.
  - Destino de tipo *Escalera*: una escalera sólo puede ser accedida directamente desde un pasillo.
- Origen de tipo *Estancia*:
  - Destino de tipo *Estancia*: una estancia no puede ser accedida directamente desde otra porque necesariamente tiene que existir un pasillo que las comunique.
  - Destino de tipo *Pasillo*: un pasillo puede ser accedido directamente desde una estancia.
  - Destino de tipo *Escalera*: una escalera no puede ser accedida directamente desde una estancia porque antes se debe atravesar un pasillo.

Para todos estos casos, se debe cumplir que las dos zonas evaluadas deben pertenecer a la misma planta y tener la misma orientación para que sean vecinas. Además, aparte de guardar su propio tipo de lugar, cada cubo almacena el sitio al que da acceso. Por ejemplo, los *cubos de posición* de la cafetería de la facultad tienen como cubos de adyacencia los cubos de los pasillos norte y sur de la planta baja.

7.3.2. Dijkstra aplicado a estructuras de edificios

El Algoritmo 1 representa el algoritmo Dijkstra descrito en pseudocódigo. En la fase inicial del algoritmo, se puede ver en el bucle for de las líneas 5-8, se inicializa el conjunto de vértices que conforman el grafo y el resultado que contendrá los nodos previos de la ruta óptima hasta el origen. En este modelo el conjunto de nodos se representan con una estructura, formada por un array más un contador, que almacena los cubos de posición y el resultado que contiene la ruta se trata de la lista de cubos de posición ordenados de origen a destino. A diferencia del algoritmo genérico, en éste también se inicializan todas las distancias de cada cubo de posición a infinito ya que no se conocen las distancias entre los cubos antes de ejecutar la aplicación.

Algorithm 1 Algoritmo de Djikstra

```
1:  function Dijkstra(graph, source) :
2:
3:      create vertex set Q
4:
5:      for each vertex v in graph :
6:          dist[v] ← INFINITY
7:          prev[v] ← UNDEFINED
8:          add v to Q
9:
10:     dist[source] ← 0
11:
12:     while Q is not empty :
13:         u ← vertex in Q with min dist[u]
14:         remove u from Q
15:
16:         for each neighbor v of u :
17:             alt ← dist[u] + length(u , v)
18:             if alt < dist[v] :
19:                 dist[v] ← alt
20:                 prev[v] ← u
21:
        return false
```

Algorithm 2 Mejora para hallar el camino mínimo hasta un nodo destino

```
1: S ← empty sequence
2: u ← target
3: while prev[u] is defined :
4:     insert u at the beginning of S
5:     u ← prev[u]
6: insert u at the beginning of S
```

La fase de desarrollo del algoritmo comienza por tratar el caso base de la línea 10 en el que la distancia del *cubo de posición* origen hasta el mismo es cero. En este modelo, el tratamiento de cada nodo en la línea 12 se corresponde con cada cubo de posición comprobando si la lista de cubos de posición sigue vacía. Si no lo está entonces se procede a evaluar el cubo cuya distancia al cubo anterior sea la mínima y además se encuentre aún dentro de la lista. La distancia se calcula en unidades de *Unity* que se obtienen en tiempo de ejecución.

El planteamiento del algoritmo original está pensado para que dado un nodo origen se calculen los caminos que puede haber a todos los nodos, pero en nuestro caso nos interesa calcular desde un nodo origen la ruta mínima a un objetivo de todos los posibles caminos. En el Algoritmo 2 se plantea la mejora que permite obtener este resultado. La ruta final óptima se calcula recorriendo la lista de nodos de forma inversa (de destino a origen).

Algorithm 3 Djikstra aplicado al modelo

```
1:  procedure CaminoMinimo(adyacencia, listaCubos, origen, destino) :
2:
3:      crear array de cubos A
4:      contador ← 0
5:
6:      for each cubo c in listaCubos :
7:          A[contador] ← c
8:          A[c].distancia ← INFINITO
9:          ruta[c] ← NULL
10:         contador ← contador + 1
11:
12:     A[origen].distancia ← 0
13:
14:     while A no es vacio :
15:         u ← cubo en A con minima distancia a A[c]
16:         if (u == cuboDestino) :
17:             rutaOptima ← lista vacía
18:             u ← destino
19:
20:             while ruta[u] esta definida :
21:                 insertar u al principio de rutaOptima
22:                 u ← ruta[u]
23:             insertar u al principio de rutaOptima
24:
25:         else
26:             borrar u de A
27:             contador ← contador − 1
28:
29:             for each vecino c de u :
30:                 distanciaTotal ← A[u].distancia + adyacencia.distancia(u , c)
31:                 if distanciaTotal < A[c] :
32:                     A[c] ← distanciaTotal
33:                     ruta[c] ← u
34:
```

Una vez obtenido se comprueba si se trata del *cubo de posición* destino, en cuyo caso terminaría la ejecución aplicando la mejora del algoritmo, o si por el contrario sigue siendo parte de la ruta, donde se procedería a borrarlo y evaluar todos sus vecinos definidos en la *matriz de adyacencia*. A la distancia acumulada de la ruta óptima hasta el momento, se suma la distancia mínima obtenida hasta un cubo de posición vecino.

El algoritmo que hemos aplicado en el modelo de la facultad y que sirve también para otros edificios, basado en cubos de posición para el cálculo de la ruta mínima se puede ver en el Algoritmo 3.

7.3.3. Verificación del algoritmo

Ante la complejidad que presenta el algoritmo, se exponen los siguientes casos que verifican su correcto funcionamiento. Por la explicación anterior, se entiende se calcularán todas las rutas posibles desde el origen hasta alcanzar la ruta óptima. Por ello, se explicará el desarrollo del algoritmo para conseguir exclusivamente el camino mínimo. Todos los casos se ilustrarán con una plantilla como la Figura 7.1, que se corresponde con una proyección horizontal [1] de la planta baja de la facultad.

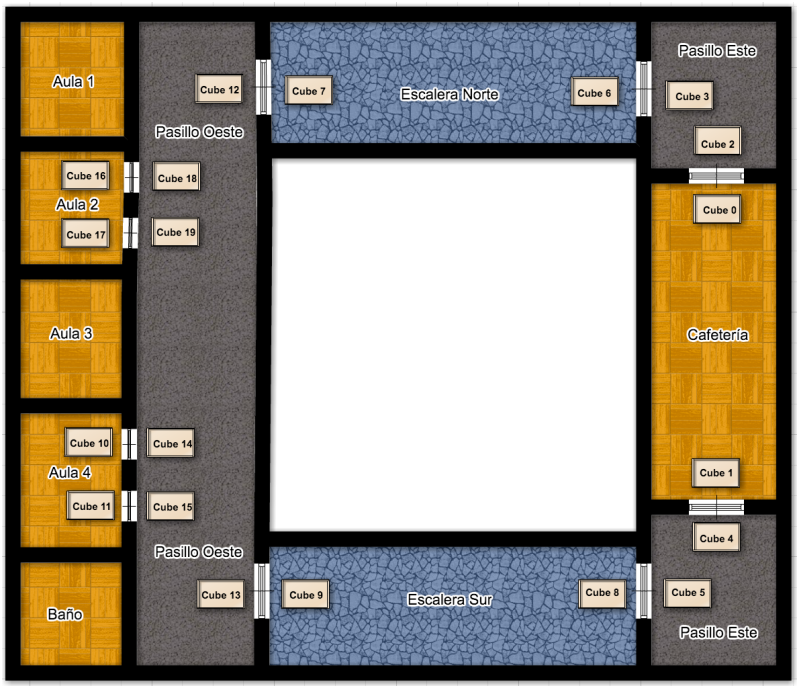


Figura 7.1: Plantilla de la planta baja

[1] Representación gráfica bidimensional de un proyecto, ubicación y dimensiones, o partes del mismo sobre un plano horizontal visto desde arriba.

7.3.3.1. Caso base

En la situación de que sólo existiese un *cubo de historia*, el camino mínimo estaría formado por el *cubo de posición* inicial ya que no se puede dar el caso de que la ruta sea vacía y además el origen se corresponde con el destino. En la Imagen 7.2 se ve un ejemplo, en el que el jugador está situado en el cubo `Cube1` y hay un único *cubo de historia* situado en la mitad de la cafetería. Tras empezar la ejecución se muestra por consola la carga de la *matriz de adyacencia* y el resultado obtenido después de realizar el algoritmo de *Dijkstra*. En este ejemplo el camino mínimo es equivalente a {`Cube1`}.

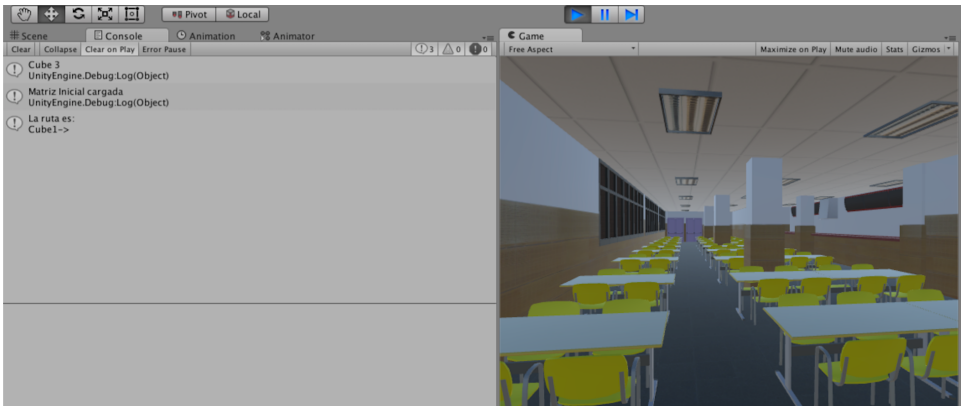


Figura 7.2: Caso base

7.3.3.2. Caso óptimo

El recorrido consiste en ir desde el aula 4(`Cube11`) a la cafetería (como se puede ver en la Figura 7.3). El algoritmo empieza buscando los cubos vecinos del origen obteniendo `Cube10` y `Cube15`. Escoge el más cercano a la estancia que resulta ser el `Cube15`, el cual está relacionado con todos los cubos del pasillo oeste más el `Cube11`. Por lo que ahora el algoritmo buscará entre todos los nuevos vecinos el más cercano a la cafetería siendo el `Cube13`. Llegados a este punto, éste tiene asignado como lugar adyacente la escalera sur(entrada `Cube9`) que será el siguiente punto por el que continuar. Se vuelve a realizar el proceso seleccionando esta vez `Cube8`, cuyo lugar adyacente es un pasillo. Se cambia de orientación y se escoge `Cube5` como el vecino más cercano (perteneciente al pasillo este). Se repite el proceso optando por el `Cube4`, siendo el siguiente `Cube1` que es el último nodo de la ruta para llegar al objetivo `Cube0`.

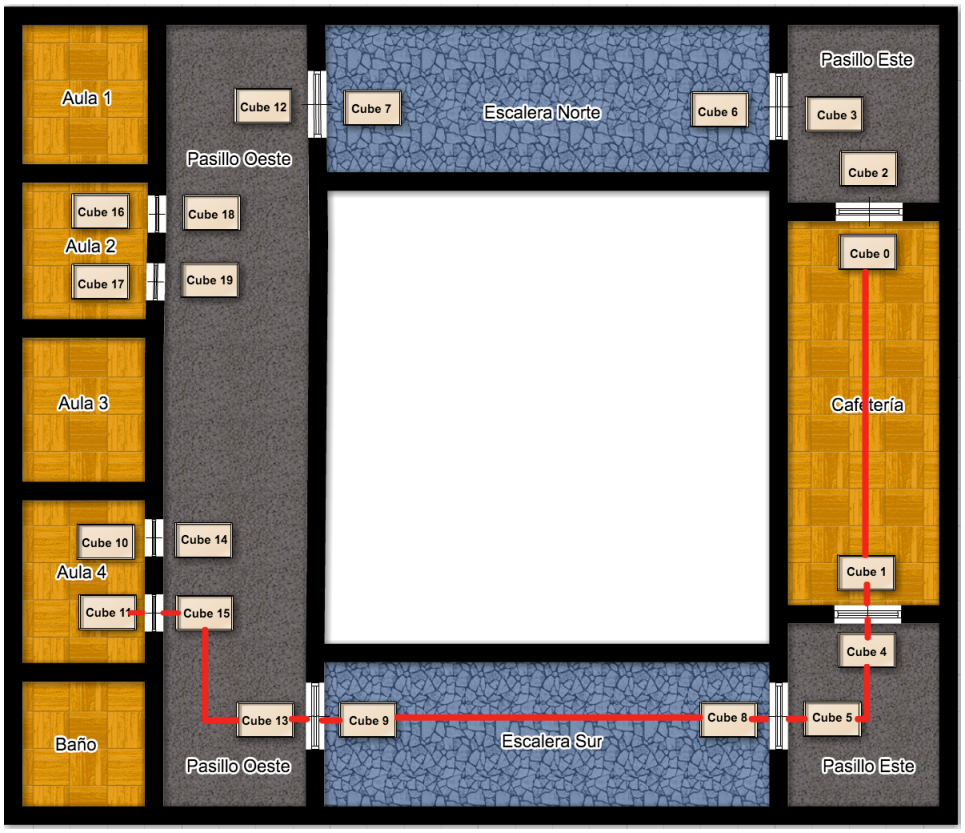


Figura 7.3: Caso óptimo

7.3.3.3. Caso de desviación

En este caso el jugador no sigue el camino óptimo inicial sino que decide hacer caso omiso a las indicaciones y desviarse alejándose del destino lo máximo posible. A continuación se ilustrará a través de los planos de la planta baja un ejemplo en el que el algoritmo va recalculando la ruta óptima y cómo afectan las decisiones que toma el jugador.

Suponemos que el jugador tiene asignado como cubo de posición origen `Cube1` y como cubo de posición destino `Cube0`, ambos situados en el interior de la cafetería. La ruta óptima que tiene que seguir aparece en la Figura 7.5 y la leyenda aplicada a todas las figuras en la Figura 7.4.

El jugador se desvía y sale de la cafetería en dirección contraria al destino accediendo al pasillo este. Avanza por la escalera sur sin subir a la planta de arriba hasta llegar al pasillo oeste. Cuando se encuentra en el pasillo continúa en dirección a la escalera norte hasta cruzar la mitad del pasillo como se ve

en las imágenes sucesivas. Entra en el cubo de posición `Cube19` (Figura 7.11) y se recalcula una nueva ruta óptima diferente a las anteriores ya que esta vez el camino más corto para llegar al `Cube0` es a través de la escalera norte.

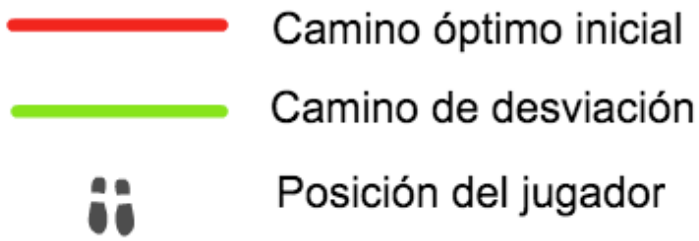


Figura 7.4: Leyenda

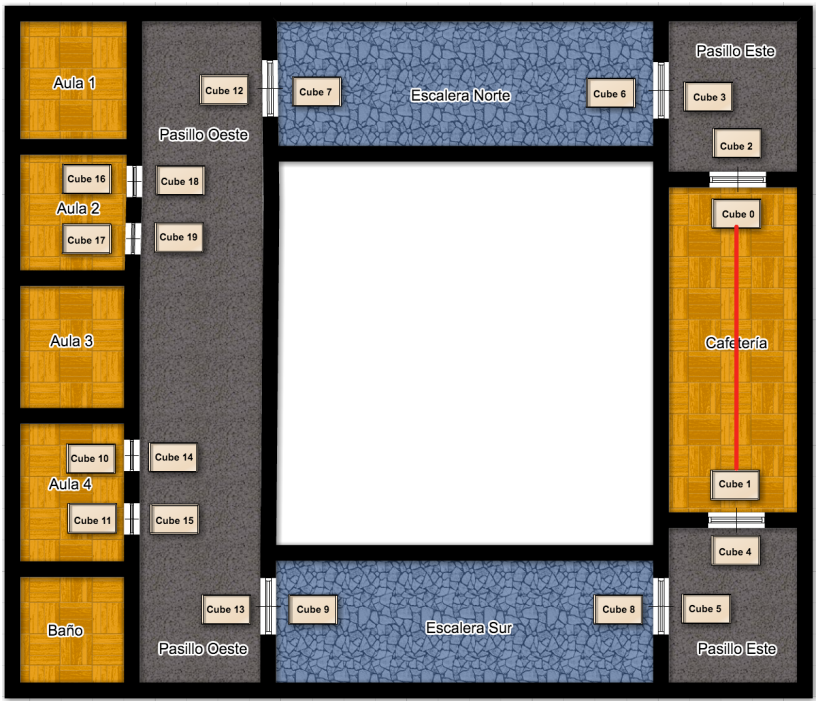


Figura 7.5: Ruta ótima inicial. Ruta: {Cube1, Cube0}.

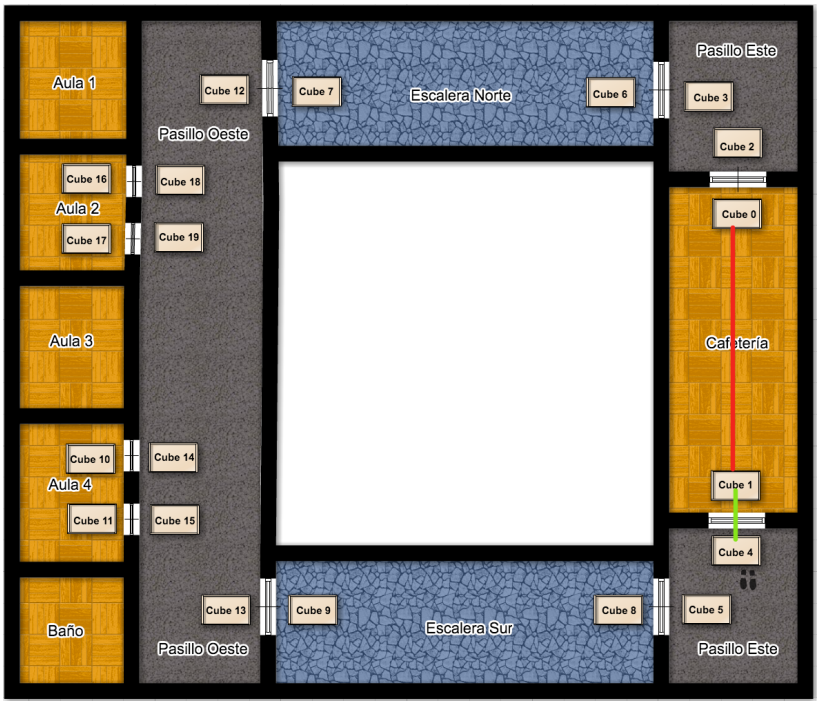


Figura 7.6: Ruta: {Cube4, Cube1, Cube0}.

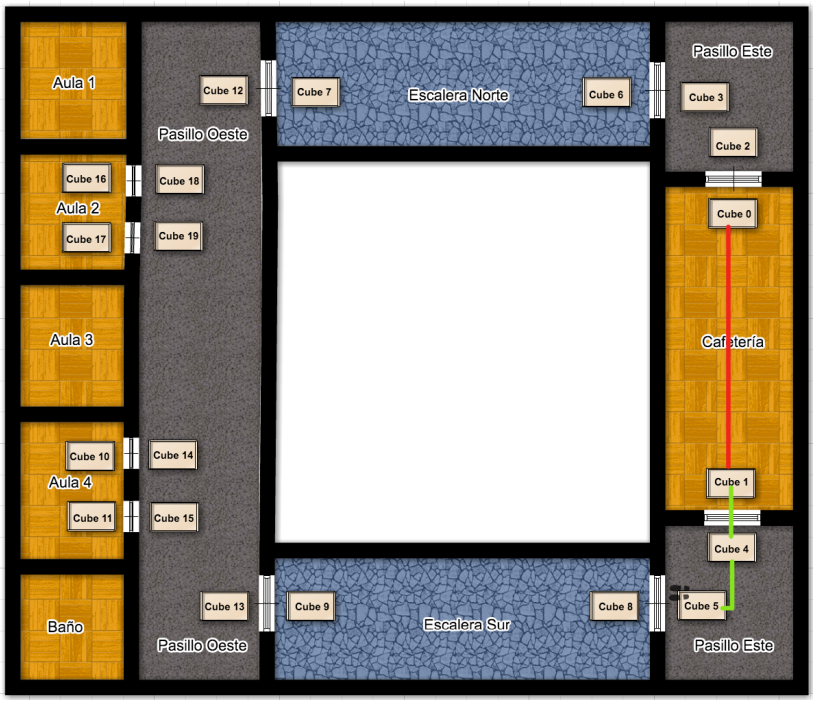


Figura 7.7: Ruta: {Cube5, Cube4, Cube1, Cube0}.

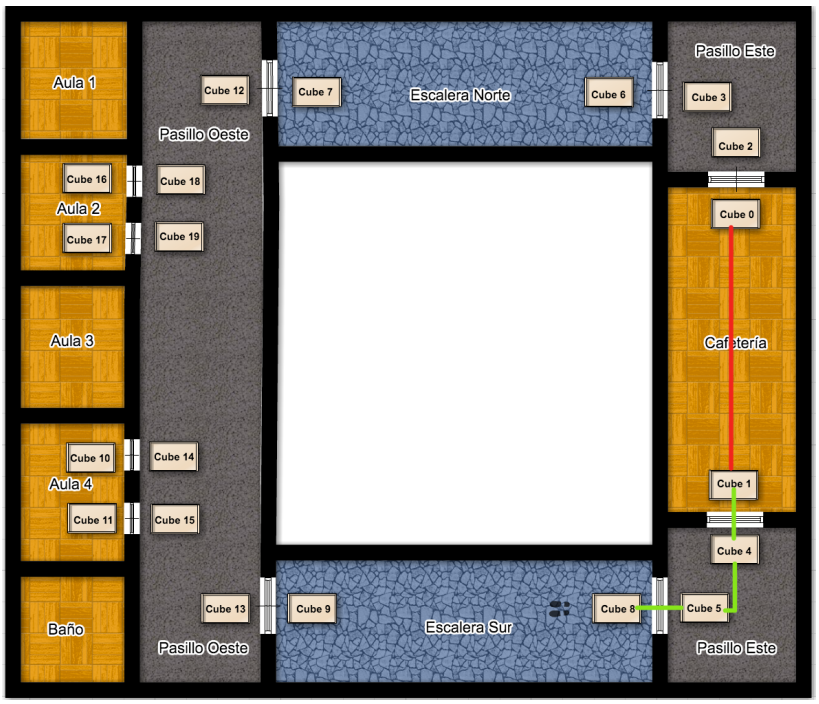


Figura 7.8: Ruta: {Cube8, Cube5, Cube4, Cube1, Cube0}.

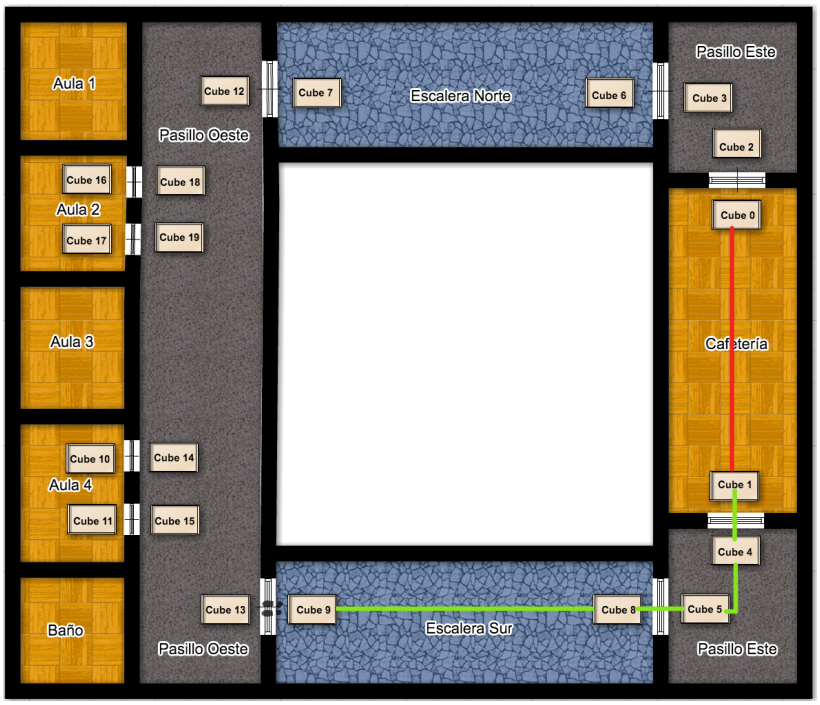


Figura 7.9: Ruta: {Cube9, Cube8, Cube5, Cube4, Cube1, Cube0}.

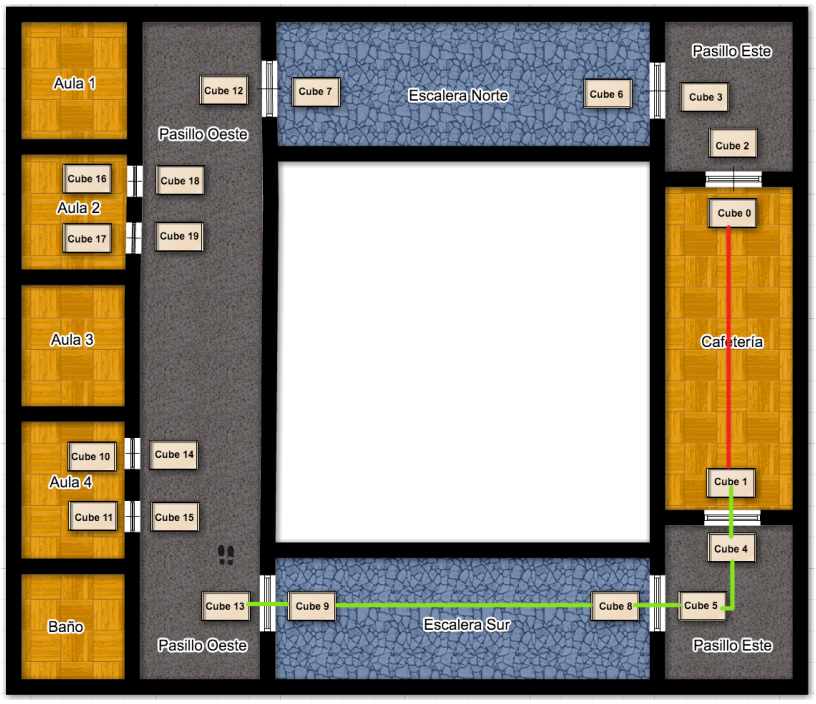


Figura 7.10: Ruta: {Cube13, Cube9, Cube8, Cube5, Cube4, Cube1, Cube0}.

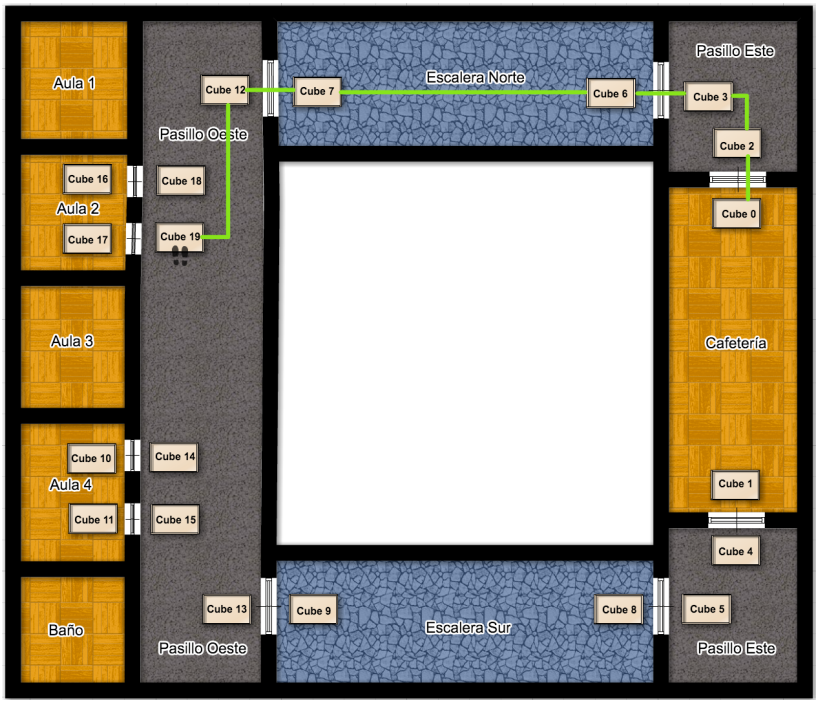


Figura 7.11: Ruta: {Cube19, Cube12, Cube7, Cube6, Cube3, Cube2, Cube0}.

7.3.4. Integración del algoritmo en el funcionamiento de las historias

El primer paso para vincular las historias con el algoritmo es determinar si existe una relación entre los *cubos de posición* y los *cubos de historia*.

A través de los *cubos de historia* podemos conocer los lugares a los que tiene que dirigirse el jugador, pero no es posible conocer la ruta que tiene que tomar basándose únicamente en ellos. Esto es debido a que la *matriz de adyacencia* sólo utiliza los *cubos de posición* para establecer la relación de vecindad. Entonces, como los lugares en los que se encuentran todos los *cubos de historia* son conocidos, basta encontrar los pertenecientes a los *cubos de posición* para calcular el camino mínimo.

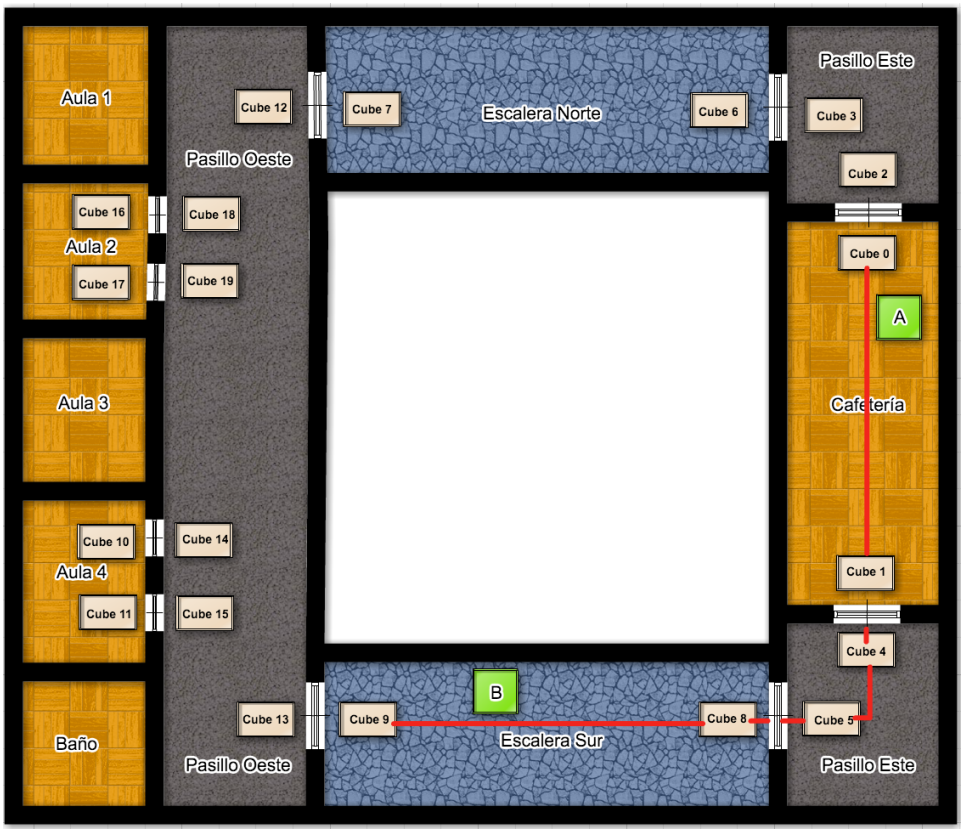


Figura 7.12: Camino desde el cubo de historia A al B utilizando el algoritmo

Por ejemplo, el jugador acaba de completar un objetivo (`A`) que se encuentra en la cafetería de la facultad y se dirige al siguiente (`B`) que se encuentra en la escalera sur de la planta baja. Siguiendo el plano de la Figura 7.12, el

cubo de posición origen del camino mínimo será uno de los que se encuentren dentro de la cafetería (`Cube0` o `Cube1`). Del mismo modo, el *cubo de posición* destino para llegar a la escalera sur será a través de uno de las dos entradas (`Cube8` o `Cube9`). En el caso de disponer de más de un *cubo de posición* para identificar el lugar, se seleccionará el más próximo al *cubo de historia* del objetivo.

El camino mínimo obtenido es { `Cube0, Cube1, Cube4, Cube5, Cube8, Cube9` } siendo el origen y destino `Cube0` y `Cube9`, respectivamente.

Por fin conseguimos dar con una solución que nos permitía saber si el jugador decidía cambiar de rumbo como se puede ver en la sección 7.3.3.3, pudiendo gestionar los errores por medio de mensajes que veremos en el próximo capítulo.

Capítulo 8

Text To Speech (TTS)

La voz de la belleza habla en voz baja, solamente se arrastra en las almas más despiertas.
Friedrich Nietzsche

8.1. Introducción

Siguiendo los pasos de Stanley Parable, la interacción con el jugador supone integrarle, transmitirle y volcarle completamente en cada historia para que pueda disfrutar de toda una experiencia de juego dentro de una aventura conversacional. La parte visual es uno de los factores más relevantes a la hora de cumplir estos parámetros, pero falta un aspecto aún sin tratar: la voz, la narración, el sonido de un relator que contagie al jugador con sus palabras, que lo mantenga expectante a que llegue el siguiente punto de la historia. En este capítulo nos centraremos en cómo conseguimos introducir la voz artificial que permitirá la conversión de los textos que se mostraban por pantalla.

8.2. Grabación y reproducción de audio

En un primer momento intentamos introducir nuestras voces para dar vida al texto de las historias, probándolo por primera vez en el comienzo del Minijuego 5.6 utilizando un sintetizador que robotizaba nuestra voz. Aunque el resultado obtenido era bueno y se podía dotar con emociones como si del doblaje de una película se tratase, se encontraron una serie de problemas que nos llevó a descartar este planteamiento.

El principal inconveniente suponía no poder hacer la conversión de texto a voz de forma dinámica, lo que obligaba a que estuviese todo predefinido para cargar los archivos de audio necesarios. Es decir, si queremos contar una aventura que tiene cincuenta objetivos con distintos textos, necesitamos un total de cincuenta clips de audio con las frases previamente grabadas para después poder reproducirlas. De otra forma, es posible llegar a un nivel de composición de frases más profundo si se graban diversos archivos de audio que contengan palabras, pero resultaría mucho más costoso. Al estar realizando una implementación cuyo principal objetivo es la generalización de las historias y que éstas se narren en tiempo real, no podíamos utilizar estos métodos.

8.3. Procedimientos para la implementación del sistema de sonido

Extrapolando la idea de introducir un sistema capaz de generar voz a partir de texto (Dutoit, 2013), empezamos buscando las diferentes posibilidades para integrarlo en `Unity`. Conviene destacar que el sistema operativo utilizado para la implementación de todo el proyecto está realizado en OS X, lo que supone un factor importante en esta sección como veremos más adelante. A continuación exponemos los tres procedimientos que se consideraron.

- *Google Text To Speech*
- *Easy TTS*
- *Comando de terminal*

8.3.1. Google Text To Speech

Es una herramienta que permite la traducción de una frase para poder escucharla y aprender así cómo se pronuncia, debido a la necesidad de muchos usuarios para poder comunicarse. Desde el punto de vista de nuestro proyecto no resulta imprescindible pero sí interesante para lo que puede aportar.

Al principio este sistema sólo era posible a través del soporte de *HTML5* de los navegadores accediendo por la url. Por ejemplo:

http://translate.google.com/translate_tts?tl=es&q=esto es una prueba

A muchos programadores les resultó interesante usar este servicio para que más adelante se consiguiera utilizarlo desde código, siendo una de las mejores alternativas para no tener que implementar un *TTS* o tener que pagar por uno.

El funcionamiento aplicado en el motor `Unity` se basa en establecer la conexión con el servidor y a su vez se le asigna a la url el fragmento de texto que se desea traducir. Después mediante un `audio.clip` se especifica el formato de audio a convertir y se termina ejecutando `audio.play` para reproducirlo. Uno de los principales inconvenientes de este sistema es la necesidad de disponer de una conexión a internet para poder realizar dicho proceso.

Fue la primera opción que escogimos para implementar, pero rápidamente desistimos en la idea al darnos cuenta de que en la última versión que daba soporte *Google* establecía el test CAPTCHA *(Completely Automated Public Turing test to tell Computers and Humans Apart)*.[1] Esto impedía la conexión con el servicio, dando lugar a una *API* de pago que cumple dicha funcionalidad. Su utilización supone un coste en relación al número de caracteres de texto traducidos.

8.3.2. Easy TTS

Es un plug-in que permite utilizar fácilmente la función nativa de conversión de texto a voz de *iOS* y *Android*. Está disponible en la tienda de *Assets de Unity* por un módico precio.

Después de un análisis exhaustivo, la forma de poder integrar un *TTS* en un sistema operativo diferente de *Windows* como es en el caso de *OS X*, resultó extremadamente complicado.

Este plug-in parecía una buena forma para poder implementar la funcionalidad deseada, además era compatible con nuestro sistema operativo. Llegando a ser la solución a nuestro problema, antes de pagarlo investigamos y nos aseguramos de que realmente merecía la pena. Por lo que observamos las opiniones de otros clientes que habían usado el servicio dándonos unas buena impresión.

Estábamos dispuestos a adquirirlo, pero nuestros tutores nos dieron una alternativa mucho mejor que podíamos implementar y que abarcaría toda la funcionalidad que ofrecía este plug-in.

8.3.3. Comando de terminal

La última y mejor opción no consiste en usar un programa de pago, ni un servicio que requiera conexión a internet o cualquier recurso que pueda suponer un problema. Es una alternativa que únicamente usa recursos propios del sistema nativo *OS X*, en este caso los comandos de la terminal. Concretamente

[1] Se trata de una prueba desafío-respuesta utilizada en computación para determinar cuándo el usuario es o no humano.

el comando `Say` que actúa de *TTS* convirtiendo texto a voz con múltiples opciones, que permiten tanto elegir el dispositivo por el que reproducir el texto como el archivo de audio en el que guardar la voz. Su principal uso en este sistema operativo se da en la aplicación *Dictado* aunque también puede ser usada desde la terminal.

La única desventaja que tiene esta alternativa es la incompatibilidad con *Windows*, ya que el comando *Say* sólo está disponible para el sistema en el que se desarrolla. Es cierto que existe una gran cantidad de software que permite realizar la función de *TTS* en *Windows*, pero uno de los objetivos esenciales de este proyecto es reducir la carga de trabajo del usuario para que pueda crear una historia. Por consiguiente se estudiaron diferentes alternativas. Una buena solución consistía en usar el espacio de nombres `System.Speech.Synthesis`, el cual nos proporcionaba una interfaz que podíamos modelar mediante un *plugin*(`.dll`). Aunque ésta también consideraba usar recursos externos, era más factible porque al menos estaba diseñado por nosotros. Sin embargo, desistimos en esta idea cuando nos dimos cuenta que la librería sólo estaba disponible para *.NET* [2] y además era incompatible con el entorno de programación *MonoDevelop* con el que se trabaja.

Finalmente, la solución llevada a cabo en el proyecto para las plataformas *Windows* es omitir la voz del narrador ya que desde *OS X* no se puede integrar bibliotecas de otros sistemas operativos para ejecutar código específico de dicho sistema. En su lugar, se asigna un tiempo estimado de duración de la aparición del mensaje que depende de su longitud. Además es necesario conocer el intervalo de tiempo en el que tiene que ejecutarse el mensaje. Se toman como referencias el momento en el que el usuario comienza a jugar (mediante la variable del sistema `Time.realtimeSinceStartup`) y el instante en el que se le empieza a mostrar el mensaje por la pantalla obteniendo la diferencia que existe entre ellos.

Afortunadamente, *Unity* integra una serie de directivas de precompilación mediante las cuales podemos diferenciar entre distintos sistemas operativos y versiones, consiguiendo interoperar entre múltiples plataformas y permitiendo cambiar fácilmente entre ellas. La implementación para dicha diferenciación se lleva a cabo mediante secuencias de comandos para compilar y ejecutar una sección de código exclusivo para cada una de las plataformas soportadas.

Las instrucciones comparativas resultan útiles cuando utilizan estos parámetros para definir cada tipo como se puede ver en la Figura 8.1. Entre las diversas directivas que ofrece el entorno se encuentran tanto para las plataformas más conocidas (*Android*, *IOS* o *Linux*) como para las menos

[2] Proporciona un modelo de programación completo para la creación de todo tipo de aplicaciones muy similar a C#.

(*SamsungTv* o *Tizen*). Además, *Unity* es capaz de detectar en tiempo real el tipo de directiva de preprocesamiento aplicada, inutilizando la parte de código incompatible con el sistema operativo utilizado por la máquina.

```
#if UNITY_EDITOR_WIN
UnityEngine.Debug.Log ("windows");
#else
UnityEngine.Debug.Log("OS X");
#endif
```

Figura 8.1: Directivas de preprocesamiento

8.4. Sistema de narración por voz

Volviendo la vista atrás, en el Capítulo 6.3 de mensajes surgió el problema del solapamiento de mensajes al que no se llegó a dar una solución. Se planteó la posibilidad de que su duración de aparición en la historia dependiese de la longitud de los mismos. En este punto llegamos a la conclusión de que no resultaba una buena solución debido a la inexactitud con la que se mide el tiempo de aparición del mensaje en función de sus caracteres. La forma más efectiva para conseguir una narración coherente del texto es establecer el tiempo en función de lo que tarde la voz del *TTS* en relatar cada mensaje. Por tanto, en esta subsección explicaremos la forma en la que se resuelve este problema sin considerar la longitud del texto.

8.4.1. Acumulación de mensajes

El modo de interacción de *Unity Engine* con la terminal se realiza con la clase `Process` que utiliza la librería `System.Diagnostics` que proporciona acceso a procesos locales y remotos, permitiendo iniciar y detener procesos del sistema local. Dado que la forma de implementar el *TTS* es mediante el comando *Say*, es conveniente llevar a cabo un control sobre la gestión de los procesos que ejecutarán dicho comando durante el desarrollo del juego. El objetivo es conseguir que la voz del *TTS* narre el mensaje que se acompañará al comando en forma de argumento. De esta forma, utilizaremos la función `Start()` a la cual le pasamos el comando que deseamos ejecutar (en este caso *Say*) más el mensaje.

Para la resolución del problema de solapamiento de mensajes se ha creado un sistema de acumulación que permite almacenarlos para poder establecer el orden en los que tienen que ser reproducidos. De esta forma, si el jugador llega a pasar por varios cubos de historia sin que el mensaje del primero haya terminado, los siguientes se reproducirán en el mismo orden en el que

entraron, es decir, sigue el concepto utilizado en estructura de datos de tipo cola, denominado *FIFO* (First-in,First-out) (Abad, 2002). El primer mensaje que se guarda es el primero que se dice siendo después eliminado de esa estructura para dar paso a los siguientes.

8.4.2. Niveles de control de comportamiento

Con el *TTS* ya integrado en el videojuego, el narrador podía llegar a dar un poco más de vida al texto con su voz, dotándole con una apariencia más humana que se acerque más a nuestra realidad. En nuestro caso, el narrador funciona como un sistema artificial que se va adaptando a la nuevas situaciones, siempre intentando hacer que el jugador recapacite en su decisión mediante frases que cada vez se van tornando en un tono más directo y, quizás en algunas ocasiones, agresivo como se vió en *The Stanley Parable* 3.3. Por todo ello una vez que ya teníamos conseguida la narración de los mensajes de historia, pasamos a implementar un algoritmo capaz de generar de forma aleatoria diferentes frases con el mismo significado. De esta forma conseguimos reducir la repetición de mensajes cada vez que se inicia el juego y se decide tomar un camino alternativo.

El objetivo de este sistema reside en la idea a la que se ha estado haciendo referencia a lo largo de todo el documento: permitir a la persona que vaya a utilizar esta estructura editar la mayor parte de aspectos del juego. Por supuesto, las intervenciones del narrador que dan vida al juego no iba a ser una excepción. Así pues, el primer paso para construir este sistema de frases es definir la estructura del archivo `mensajesControl.xml` explicada en la sección5.5. De esta manera se consigue que el usuario pueda editar cada frase del narrador, cargándose dinámicamente y sin que ninguna esté predefinida dentro de la aplicación.

La figura del narrador cuenta con una forma de expresarse dividida en tres niveles de intensidad. Cuanto más bajo sea el nivel más afable y comprensivo se mostrará con el jugador. Por el contrario, cuando decida no hacer caso de sus indicaciones, se volverá cada vez más desagradable y antipático en los niveles más altos. Para alcanzar estos últimos debe de desviarse cada vez más del camino hacia el objetivo a cumplir, moviéndose por distintos lugares.

El número de sitios por los que no tiene que pasar se van registrando y cada cierto número de ellos aumenta el nivel de intensidad. En el caso de que el jugador decida seguir correctamente la historia según se le cuenta, el número de fallos se reinicia y el narrador vuelve a tomar una actitud amistosa con él.

Como se vio anteriormente, la ruta que debe seguir el jugador se resuelve con el cálculo del camino mínimo mediante el algoritmo de *Dijkstra* explicado

en la sección 7.3. A través de este algoritmo se detecta si el jugador se aleja o se acerca más al objetivo. Si la ruta obtenida tras pasar un cubo de posición resulta tener como subconjunto la calculada anteriormente, significa que el jugador se está alejando del objetivo. Se puede ver un ejemplo en la Figura 8.2.

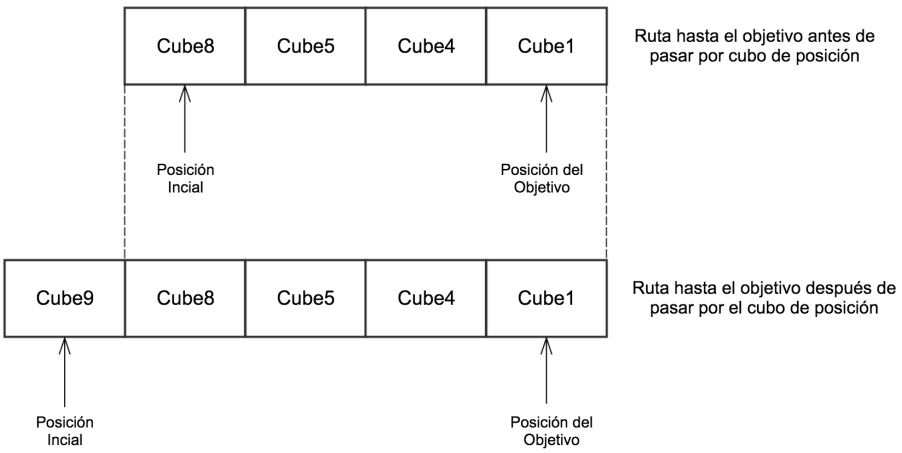


Figura 8.2: Ejemplo de un camino que es subconjunto de otro

Además, se necesita que las frases del narrador tengan sentido, se correspondan con lo que sucede en la historia en ese momento y que no sean siempre las mismas y hagan que resulte previsible y tedioso.

Por tanto, es necesario determinar cómo gestionar los niveles en los que se organizan las frases del narrador para que sean más o menos agresivas. En segundo lugar, se debe crear un sistema que defina cómo se pueden estructurar las frases de forma que, con la menor cantidad de datos posibles, se puedan generar muchas más con sentido.

El primer paso es determinar qué diferencias existen entre los niveles de las frases. En los primeros, el narrador es más respetuoso con el jugador y por ello, deberá tratarle de usted e incluso llamarle por su nombre para mantener cierta distancia entre ellos. En este caso, se ha desviado por el camino que no debería seguir pero al estar en un nivel tan bajo, implica que no ignora por completo al narrador, por lo que no hace falta indicarle el lugar al que debe regresar, sino, simplemente informarle de que no está recorriendo el camino adecuado.

Por el contrario, en los últimos niveles el narrador es menos amigable y, en consecuencia, debe tutearle y ser más directo en sus comentarios. Además, en

este momento el jugador ha llegado a un punto bastante alejado del objetivo, por lo que el narrador debe recordarle el sitio al que debe dirigirse. Si el motivo por el que le llevó a tomar ese camino fue la exploración del mismo conviene que éste pueda informarle además del lugar en el que se encuentra por si lo necesitara saber.

8.4.3. Estructuración de las frases del narrador

Tras definir la estructura de niveles, el siguiente paso consiste en generar el mayor número posible de frases. Para ello, se deben dividir en trozos con los que se puedan construir más a partir de ellos. Conviene aclarar el nivel de partición porque es posible alcanzar niveles muy complejos. Evidentemente, el menor que se puede alcanzar es a nivel de palabra, de tal forma que se establezca un vocabulario para organizar dichas frases. Pero como el tema principal de este proyecto no es la *Generación Natural del Lenguaje* (GNL) (Ballesteros, 2009), se ha decidido realizar la fracción en trozos más grandes con los que también se puede conseguir el objetivo propuesto.

Cada uno de estos tipos se puede clasificar en un nivel concreto atendiendo a su construcción y la intensidad con la que la cuenta el narrador. De esta forma, la organización de las oraciones está presente en todos los niveles del archivo `mensajesControl.xml` y se divide en tres partes:

- `Frases principales` : son aquellas que pueden formar los mensajes completos del narrador por ellas mismas, sin necesidad de aperturas y cierres.
- `Frases de apertura` : tienen un uso opcional ya que pueden comenzar o no una frase principal.
- `Frases de cierre` : al igual que las de apertura, no son imprescindibles pero pueden terminar la oración principal.

8.5. Reglas para la construcción de oraciones

Con esta división se pretenden construir tanto oraciones compuestas como simples haciendo uso de todos los tipos de frases, o solamente algunos de ellos. Siempre que los tipos usados para construir el mensaje completo se encuentren dentro del mismo nivel, se garantiza que no habrá ningún problema de coherencia en el texto mostrado en la pantalla si se siguen las reglas que se enuncian a continuación.

En el caso de usar frases simples es necesario especificarlo dentro del archivo indicando que terminan con un punto. Dentro de las aperturas y cierres,

se pueden usar oraciones interrogativas o exclamativas tales como *¡Vaya!* o *¿Sabes una cosa?*. El usuario editor deberá indicarlo de forma distinta a como se haría para una oración enunciativa (como por ejemplo, *En realidad no me importas.*) ya que difieren en el punto al terminar. El sistema lo omite en las aperturas al igual que en los cierres, debido al signo de apertura de admiración o interrogación. Además, en los cierres se debe señalar si es posible continuar con la frase principal o, por el contrario, termina, en cuyo caso se deberá comenzar por mayúscula.

Al contrario que las simples, las compuestas necesitan más restricciones para diferenciar los nexos que existen entre los tipos de frases. Se distinguen dos grupos principales que se forman con las aperturas y los cierres: con y sin nexo. Al igual que sucedía anteriormente con el punto final, surge una nueva dificultad en este tipo cuando se introducen comas. Respecto a las aperturas, como por ejemplo *En un alarde de rebeldía*, se indicará si al término de cada una es necesario colocar una coma para comenzar la siguiente frase por minúscula, dado que el mensaje completo no se trata de una oración simple. En los cierres sucede una situación parecida con la diferencia de que el nexo se encuentra al principio y no al final.

Para hacerlo más complicado todavía y permitir a los usuarios usar elementos de la historia para elaborar mensajes aún más personalizados, las reglas para construir las frases principales serán completamente diferentes. Según su definición, es capaz de constituir por sí sola un mensaje completo, por lo tanto no es necesario que el usuario precise ningún signo de puntuación como las anteriores reglas. Un ejemplo de esto es la frase *En su libre albedrío Gabi decidió cambiar de camino*. Las nuevas características van a permitir al usuario hacer construcciones que utilicen datos del juego según el momento en el que se encuentre el jugador.

En concreto, nos centraremos principalmente en su ubicación y en los objetivos que le quedan por realizar. Si pretende dar más prioridad a explorar que a seguir las indicaciones del narrador, resulta más conveniente enviarle información que le ayude a orientarse en la facultad con los sitios que visita. También puede ocurrir al revés, en cuyo caso es más acertado reconducirle al objetivo de la historia. Para aplicar este comportamiento en los mensajes de control, la frase principal debe terminar en un verbo sin hacer referencia a ningún lugar y especificar el tipo de mensaje. La aplicación se encarga de modificarla añadiendo la ubicación u objetivo debidamente para que la oración tenga sentido.

Un ejemplo que resume todas las reglas antes vistas se puede ver en la frase principal *Creo que te había dicho que tenías que ir*. Consideraremos que las directivas del usuario para los distintos tipos son sintácticamente correctas y que el lugar en el que se encuentra el personaje es la cafetería. Antes de procesar la oración, se obtiene el lugar del objetivo con los datos del

momento del juego en el que se encuentra y se integra con la frase principal. A continuación, el sistema verifica si se tiene que usar algún tipo de apertura o cierre. En tal caso, comprueba si es necesario añadir signos de puntuación y cuántos de ellos. Supondremos que el mensaje completo contiene *Tal vez no me has entendido bien, pero* como apertura y *¿Seguro que quieres seguir con este despropósito?* como cierre. Primero se agrega la apertura indicando que la frase principal debe comenzar por minúscula y se junta el resultado con el cierre. Como esta última se trata de una oración interrogativa, no se complementará con el punto inicial y final. El resultado final del mensaje es *Tal vez no me has entendido bien, pero creo que te había dicho que tenías que ir a la cafetería ¿Seguro que quieres seguir con este despropósito?.*

Capítulo 9

Personajes

La felicidad solo es real cuando se comparte.
Hacia rutas salvajes

9.1. Introducción

La soledad, ese sentimiento de vacío que nos inunda al estar solos, la carencia de compañía, el estado que deseamos evitar que muchas veces nos lleva a la melancolía. Es lo que siente ahora mismo nuestro personaje al encontrarse en una facultad desolada y sombría. En este capítulo introduciremos personajes, los cuales suplirán la falta de compañía pudiendo interactuar con ellos expectantes a lo que nos digan, que será en muchos casos relevante para el desarrollo de las historias. Además se podrán mover por la universidad y diferenciaremos entre personajes dinámicos y estáticos, dando una perspectiva completamente nueva al proyecto.

9.2. Modelización de los personajes

En el momento de introducir los personajes, nos encontramos que para seguir con la idea de que cualquier persona pueda crearlos, y generar su propia historia, se necesita establecer una serie de modelos. Estos moldes, `prefabs`, serán los que más tarde se instanciarán determinando distintos parámetros que les podrá dar el usuario.

Unity contiene una serie de librerías que nos aportan una gran variedad de objetos que podemos utilizar. Gracias a algunas de ellas, entre las que se encuentra `NewCharacterPack`, conseguimos varios prototipos de chicas y de

chicos, a parte de muchos complementos que nos ofrecen una diversidad de posibilidades a la hora de alternar su vestuario y su físico. Se encuentran desde diferentes tipos de texturas para la piel, el color del pelo y los ojos, hasta materiales que permiten cambiar el calzado, pantalones o camisetas. En la Figura 9.1 vemos un ejemplo del modelo al instanciarlo en *Unity*.

Figura 9.1: Prefab personaje

Dentro de estos `prefabs` de la librería anterior hay dos grandes grupos, cuya desigualdad está dada por la posición que ocupan. Para entenderlo, primero hay que ver que todos estos moldes son totalmente articulables, lo que significa acceder a todas las partes del cuerpo (como se puede ver en la estructura 9.2) y poder rotarlas a la posición deseada. De esta forma se adquieren múltiples posturas, siendo las dos principales estar de pie y sentado.

(a) Articulación del personaje

(b) Jerarquía de la anatomía

Figura 9.2: Estructuras y movilidad de las partes del cuerpo

Decidimos crear un total de doce modelos en estas posiciones, intentando generar una diversidad de opciones con las que el usuario sea capaz de interactuar e introducir dentro de la facultad. Seis de estos `prefabs` se corresponden con mujeres y los restantes con hombres. En ambos, cuatro de ellos permanecen de pie y los otros dos sentados. Cada uno tiene una determinada postura, algunas de las cuales son: con los brazos cruzados, con las manos en los bolsillos, con las piernas cruzadas, imitando que está apoyado, etc. El objetivo es dar la sensación de ver personajes diferentes mientras que se recorre la facultad, que junto con los objetos, harán que parezca un lugar más cercano a como sería en la realidad.

Es necesario que la ropa que lleve por defecto el `prefab` sea completamente blanca, que como veremos más adelante supondrá una parte imprescindible para que cuando se cargue en el juego aparezca correctamente.

9.3. Animación

La animación es un proceso capaz de dar movimiento a un objeto inmóvil, generando una ilusión óptica a través de la combinación de imágenes estáticas. Normalmente se suelen pasar a más de 15 frames por segundo[1] para conseguir este efecto. Es así como logramos dar dinamismo a los personajes.

[1]Es una unidad que mide las imágenes que se muestran por segundo.

Su integración en *Unity* se consigue mediante *Animator*, definida como una interfaz de control. Incorpora un controlador que define un conjunto de máquinas de estados[2], las cuales permiten crear, modificar y borrar diferentes animaciones, con sus respectivas transiciones. *Unity* suministra una gran cantidad de ellas ya predefinidas que se pueden incorporar a cualquier personaje fácilmente y por tanto no se precisa de una herramienta externa como Blender[3] para generarlas.

Al pensar en los diferentes situaciones para animar al personaje, se nos ocurrieron muchos posibles de los cuales sólo dos eran realmente necesarios para lo que queríamos implementar. Constituyen las acciones más habituales que utilizamos los seres humanos. Una es la de andar y la otra la de permanecer en reposo cuando estamos de pie. Utilizando de esta forma estos estados, denominados *Andar* e *Idle*(parado), conseguimos marcar el flujo de movimientos deseados. Ambos están relacionados por las transiciones que definen el paso de uno a otro, siendo siempre el inicial el de reposo. En la siguiente Figura 9.3 se ve la máquina de estados resultante.

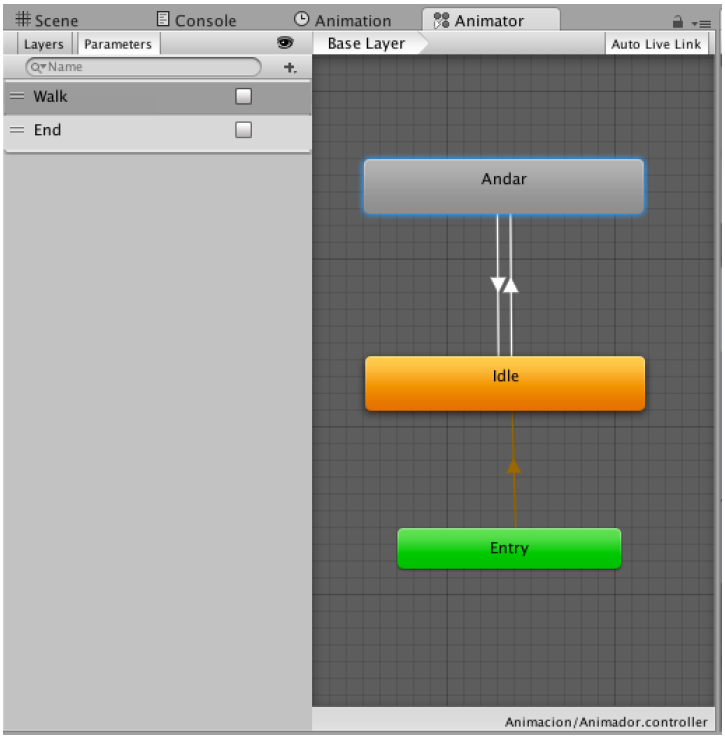


Figura 9.3: Máquina de estados

[2]Es una estructura que sirve para determinar el comportamiento de un sistema.

[3]Es un programa informático multiplataforma, dedicado especialmente al modelado, iluminación, renderizado, animación y creación de gráficos tridimensionales (Wikipedia, Blender)

Todos los personajes contienen, en los archivos *protagonistas.xml* y *personajes.xml*, la información referente a si va a estar animado o no en el juego (como se vio en la sección 5.5). Además, aquellos `prefabs` que permitan la animación de los personajes tienen asignado un `script` de comportamiento. Se debe diferenciar entre aquellos que no tienen ningún objetivo a seguir y los que lo tienen. A estos últimos, se les asigna una estructura de datos que almacena los lugares por los que tendrán que ir, por lo que, cuando se cargan en el juego, el `script` mencionado anteriormente activa la transición que hay de *Idle* a *Andar*, iniciando así la animación *HumanoidWalk*. Una vez alcanzado el último punto de la lista de objetivos, se vuelve a realizar el cambio de estado y se queda en *Idle*.

Una de las dificultades con las que se tuvo que lidiar fue la pérdida de orientación del personaje sobre el eje y. En otras palabras, cuando empezaba a andar, parecía que flotaba sobre el aire ascendiendo cada vez más, alejándose del plano del suelo. Se fijó dicho eje del personaje para que fuera constante, logrando que caminara de forma normal.

9.4. Tipos de personajes

La mayoría de los videojuegos contienen una serie de personajes clasificados según la relevancia que tienen en la trama. Son una pieza fundamental mediante la cual se obtienen diferentes roles con los que integrar al jugador cada vez más en la historia. Los más frecuentes son:

- `Principal`: es el que mayor grado de importancia tiene, sobre el que se desarrollan casi todas las historias.
- `Secundario`: es aquel que, en determinados momentos, sustenta al protagonista y puede ayudarle en sus aventuras.
- `Portavoz`: tiene un peso relevante durante el desarrollo del juego, ya que en ocasiones puede ejercer de narrador omnipresente e indicar al usuario la acción a desempeñar o seguir.
- `Estático`: no sufren ninguna evolución a lo largo de la historia.
- `Dinámico`: al contrario que el estático, presentan una transformación.

Se planteó cuáles de ellos integrar y la forma en que afectaría implementarlos. Nos interesaba simular una facultad habitada de estudiantes para que pareciese lo más real posible. Por eso necesitábamos un tipo de personaje estático [4] con el que se consigue dar el aspecto de un edificio ocupado. Por

[4]Es aquel que se encuentra sentado o de pie en las clases, laboratorios y cafetería sin hacer nada.

otro lado queríamos delegar el narrador a un personaje para llegar de una forma más cercana al jugador, lo que también supone una mayor interacción con el usuario.

9.5. Personajes de relleno

Este tipo de personaje se corresponde con el estático, el cual, aunque no tiene ningún tipo de repercusión en la historia, sí que es sumamente relevante para dar vida al juego. Su carga se produce desde el archivo `personajes.xml`. El usuario podrá decidir el nombre que le quiera dar, junto con la posición y la planta que ocupará una vez que se instancie.

Constituyen la mayor parte de los personajes que se van a introducir. Suponen una gran cantidad de espacio en memoria, por lo que, para reducir este coste, se decidió hacer su carga dependiendo de la planta en la que se encuentre el jugador. De esta forma, si consideramos que el usuario se encuentra en la primera planta, sólo estarán los que contengan el mismo identificador de planta que el personaje. Si decide cambiar de planta, se borrarán los de la primera y se instanciarán los de la nueva.

9.6. Personaje portavoz

Este personaje proporciona un inciso en la aventura, un momento donde el jugador deja de tener el control y se ve embaucado por un elemento externo que le incita a estar atento. Se encargará de ejercer de narrador en ocasiones, dando un mensaje descriptivo de lo siguiente que tiene que hacer el usuario o meramente un comentario para amenizar la historia. En otras ejercerá de protagonista secundario revelándonos algún incidente que le haya ocurrido.

Su materialización en el juego se realiza a través del fichero *protagonistas.xml*. En él, encontramos el identificador de la historia junto con el objetivo que nos permite conocer el momento en el que debe aparecer. Conociendo el cuándo, lo siguiente que necesitamos es el dónde, definido por la posición `Vector3`. También cuenta con el mensaje que debe darle al jugador, ejerciendo de portavoz como su propio nombre indica.

La perspectiva, los puntos de vista y los diferentes planos permiten hacer énfasis en diferentes escenas, personajes y acciones que de otra manera no sería posible observar, provocando un cambio que sirve para focalizar en lo que nos interesa cambiando la forma de ver las cosas. Por ello decidimos que una buena forma de introducir un personaje que influyera en la historia, y a su vez conseguir que el usuario fuese hacia él, lo mejor era hacer un cambio

de cámara. Así conseguimos captar su interés e introducirle en una nueva situación ofreciéndole otra vista.

Nos pareció conveniente dejar en manos del usuario la elección de realizar este cambio, para que tenga la mayor libertad posible a la hora de crear las historias. Por esta razón se añadió un campo que permite activar y desactivar la cámara. De este modo, si está activo, en cuanto tocamos el cubo de historia que carga el personaje se produce este efecto. Por el contrario, si se encuentra desactivado no se producirá ninguna alteración.

Es necesario crear una nueva cámara en el entorno (como se ve en la Figura 9.4) para realizar la idea descrita anteriormente. Tiene asociado un `script` que sigue los movimientos del personaje portavoz a través de la distancia y el ángulo de visión. Sin embargo, no es la única cámara con que contamos ya que, por defecto, el `FPSController`[5] del jugador lleva incorporada una. Por tanto, se debe realizar la gestión de ambas mediante un controlador que deshabilite la cámara principal, denominada `Main Camera`, dando paso a la nueva.

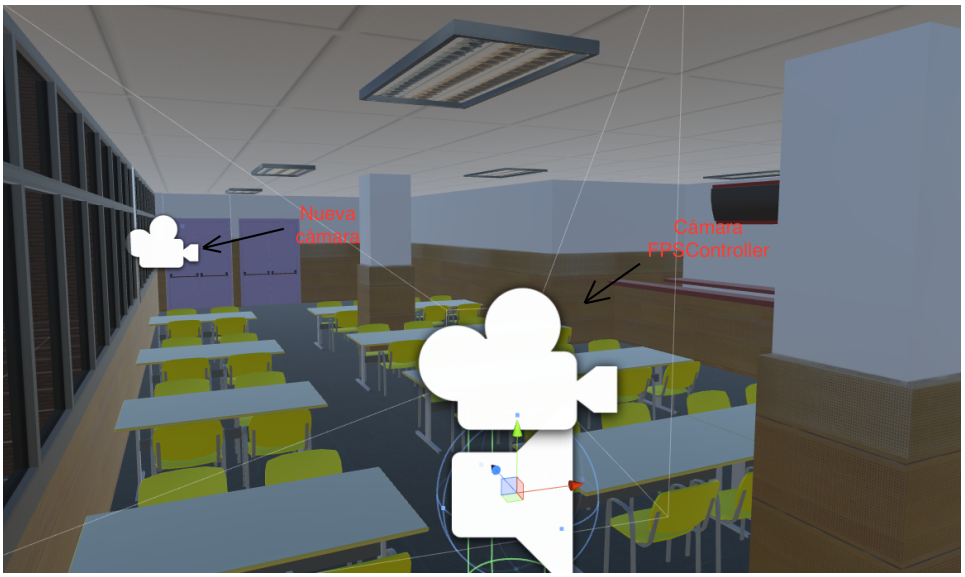


Figura 9.4: Escena con la nueva cámara

[5]Es un objeto que es utilizado como primera persona, es decir, ejercerá nuestro papel y lo controlaremos e indicaremos a lo largo del juego.

Si está activada, la duración de la toma [6] de la nueva cámara depende de dos factores que deben cumplirse:

- `Movimiento`: mientras que el personaje portavoz se esté moviendo y no llegue al destino marcado, la cámara le seguirá hasta que se pare.
- `Mensaje del narrador`: hasta que el mensaje del cubo de historia tocado por el jugador no termine, permanecerá activa.

Por lo tanto, estas dos condiciones serán las que marquen el comienzo y fin del tiempo del nuevo plano. A la vez que este cambio se produce, se deshabilita el `script FirstPersonController`[7] para impedir que el jugador siga avanzando en la historia. Así se consigue invalidar cualquier movimiento que pueda hacer y, por tanto, prevenir el conflicto con otros cubos de historia. En consecuencia, se alcanza el otro objetivo planteado que es el de atraer su atención.

El establecimiento de la comunicación con el jugador, una vez cargado y terminado el cambio de cámara, se da por la distancia. Una vez instanciado el personaje tenemos que acercarnos hasta él para que se inicie la interacción, el cual girará hacia el lugar donde nos encontramos sin importar la posición en la que esté, fijando así el inicio del diálogo que permitirá mantener una conversación cara a cara. Con el objetivo de evitar la distracción del jugador se le priva de los controles al igual que en el caso anterior. Vemos un ejemplo en la siguiente Figura 9.5.

Figura 9.5: Manteniendo una conversaión con un personaje

[6] Es el fragmento que se graba desde que la cámara comienza a registrar hasta el corte de la misma.

[7] Es un controlador de primera persona con el que se puede determinar la velocidad del personaje tanto al andar como al correr, la gravedad, la longitud de salto y otros componentes propios de éste.

La destrucción del personaje se produce cuando tocamos el siguiente cubo de historia que se carga tras haber hablado con él.

9.7. Características comunes

Los dos tipos de personajes conforman un gran número de objetos que estarán dentro de la facultad. Por ello se decidió dar la opción de cambiarles los colores de la ropa, permitiendo generar una cantidad ilimitada, para no caer en la monotonía y repetición de verles siempre igual.

Para llevar a cabo este propósito se consideraron dos opciones con sus respectivas ventajas e inconvenientes. La primera consistía en darle la oportunidad al usuario de tener una gama de colores muy extensa, concretamente 16,7 millones, mediante el modelo de color *RGB*. Utiliza el rojo, verde y azul, a los cuales se les puede dar un valor entre 0 y 255 indicando la intensidad de cada uno. Este sistema es el más efectivo, ya que cubre el mayor número de colores posibles, pero por otro lado supone un alto coste de tiempo debido a que cada vez que se quiera introducir un personaje se ha de dar tres números(RGB) por cada tipo de prenda. Principalmente se puede ver en el caso de que haya un número muy elevado de personajes, teniendo que buscar para cada color el equivalente a este sistema y poner los tres valores que lo conforman. La segunda opción, aunque acota bastante este rango de colores, consiste en utilizar los que vienen preestablecidos por la clase `Color`. Aunque solamente son once, resulta más fácil asignarlos porque simplemente podemos hacer uso de su nombre (Ejemplo: `Color.gray`).

Por último, después de analizar los pros y los contras, nos decantamos por usar la segunda alternativa para los personajes de relleno que serán la mayoría, ganando más eficiencia en el momento de crearlos y para los personajes portavoz, la otra posibilidad. Como estos últimos aparecen en menor número, se les puede asignar más colores para moldearlos de la mejor forma y darles mayor visibilidad ya que tienen mayor relevancia en la historia.

Capítulo 10

Modos de juego

La única posibilidad de descubrir los límites de lo posible es aventurarse un poco más allá de ellos, hacia lo imposible.
Arthur C. Clarke

10.1. Introducción

Llegados a este punto, donde se encuentran generalizados todos los elementos necesarios para la creación de historias mediante los archivos `XML`, se reiteró en la idea inicialmente pensada de no tener que requerir al usuario la instalación del programa `Unity3D` para definir las posiciones y rotaciones tanto de personajes, cubos y objetos. Además, le resultaría más complicado, ya que necesitaría conocer el entorno a un nivel intermedio para desenvolverse bien. La intención es ayudarle y facilitarle lo máximo posible dicha tarea. Es por ello que decidimos hacer dos modos de juego: uno donde se desarrollan las historias y otro que hemos denominado *Modo Editor* que constituye una nueva escena y también nos permitirá evitarle ese requisito al usuario.

10.2. Pantalla de Inicio

En el momento de llevar a cabo la diferenciación entre modos, se necesitaba una primera pantalla inicial en la cual poder seleccionarlos. Optamos por implementar una nueva escena formada por un `Canvas` (explicado en el apartado 4.2.5) que contiene dos elementos de texto que funcionan como botones y una imagen que establece el fondo de pantalla.

Para resaltar el texto se descargó un estilo nuevo más visual `Orange_juice` (Murphy, 2005), consiguiendo diferenciarlo del que viene por defecto. Ambos botones contienen dentro de la función `on Click()`[1] un `script` que será el que cargue un modo u otro. Mientras que no se pulse ningún botón se mantendrá la pantalla de inicio (como se puede ver en la Figura 10.1).

Figura 10.1: Pantalla de inicio

10.3. Descripción de los modos de juego

Los dos modos están constituidos por el mismo escenario: la facultad. Cada uno está pensado para un tipo diferente de usuario o de rol. El de Historia ejerce de videojuego y es en el que se producirán las aventuras; el rol será de jugador. En cambio, en el *Modo Editor*, el usuario se acerca más a la parte de implementación del juego, pudiendo editar un objeto que más tarde formará parte de él. A continuación se expone la descripción y un ejemplo de ambos.

10.3.1. Modo Historia

Está compuesto por todos los elementos explicados en los capítulos anteriores, desde personajes, objetos, narrador y cubos hasta los objetivos que marcarán el desarrollo de todas las historias. Con el objetivo de unificar lo que hemos visto, se presenta un ejemplo donde trataremos y utilizaremos todos estos elementos.

[1] Evento de Unity que se activa cuando se pulsa un elemento.

10.3.1.1. Ejemplo: Época de exámenes

Se desea realizar una historia en la que partimos desde la entrada de la facultad y debemos llegar a tiempo a hacer un examen. Por otro lado, se plantea otra historia, cuyo transcurso puede ser posterior o anterior a la realización del examen. Consiste en encontrar a un profesor con el que tenemos una revisión para ver si nos la puede cambiar a otro día porque el horario en el que tiene lugar nos coincide con otro examen. Para ello, es necesario hablar con varios personajes que nos den ciertas pistas relativas a la ubicación del aula y del profesor, además de la información adicional que nos de el narrador.

Como explicamos en capítulos anteriores, el archivo de `facultad.xml` tiene todos los cubos de posición necesarios para construir una historia entre la planta baja y la primera planta. Así pues, no será necesario modificarlo porque puede que los cambios realizados hagan que el juego no funcione correctamente. A pesar de eso, el usuario lo tiene al alcance ya sea para ajustar los cubos de posición a una ubicación más cercana o lejana de las puertas según lo considere. También puede modificar aspectos que no influyan tanto en el desarrollo de las historias como puede ser el nombre de cada lugar.

Por tanto, el primer paso para crear una historia es hacerlo directamente desde el archivo `historia.xml`. Según el enunciado, el jugador parte desde la entrada de la facultad. Sin embargo, no es posible obtener esa ubicación exactamente, el usuario debe indicar que se desea partir desde una cercana a ella. Por ejemplo, la puerta que da acceso a secretaría es una de las más próximas. Una vez escogida la posición deseada procederemos a agregarla en dicho archivo con el formato que se muestra a continuación:

```
<ContenedorHistorias>
    <PosicionInicial planta="baja" lugar="pE" puerta="sNorteSecretaria"/>
    <Historias> </Historias>
</ContenedorHistorias>
```

Esta posición inicial se relacionará con el lugar asociado que se encuentra dentro del archivo `facultad.xml`. De esta forma se pueden conocer sus coordenadas para situar al personaje dentro del cubo de posición. El formato del lugar en `XML` es el siguiente:

```
<Lugar id="pE" tipo="pasillo" orientacion="E">
   <Entrada id="sNorteSecretaria" nombre="pasillo noreste de la planta baja"
        adyacente="estancia">
            <Orden>0</Orden>
            <Posicion>
                  <X>-12.499</X> <Y>0.934</Y> <Z>7.023</Z>
             </Posicion>
      </Entrada>
</Lugar>
```

Ahora, ya podemos empezar a editar los objetivos de las historias. Tanto en la primera como en la segunda, sólo queremos que el personaje vaya al lugar adecuado para realizar el examen o revisar uno anterior. Si se añaden más hitos, entonces se puede construir un camino más estricto hasta llegar al aula. Dependiendo de cómo se cree el guión, puede resultar una historia en la que el narrador marque objetivos totalmente diferentes respecto al del final o que sugiera los lugares a atravesar en todo momento. En cualquier caso, el jugador tendrá que completar el objetivo anterior para poder pasar al siguiente.

En la primera historia tendremos varios hitos que se deben completar antes de llegar a hacer el examen. Primeramente, vamos a suponer que el protagonista no sabía del control. Así que, al llegar al pasillo oeste, veremos al fondo a nuestros compañeros de clase. Seguidamente nos daremos cuenta de que es la hora del examen que pensábamos que tendría lugar la próxima semana. A continuación, nos dirigiremos a la máquina expendedora de bebidas a comprar un refresco para calmar los nervios. Al llegar a ella nos daremos cuenta que no llevamos el dinero suficiente encima, así que se lo intentaremos pedir a un amigo (que se encuentra cerca de la barra de la cafetería). Cuando hablemos con él, nos dirá que lleva lo justo para comprarse la comida, así que no nos dará nada. Tras ello, iremos directos a realizar el examen porque va a ser la hora y el profesor está a punto de llegar. Llegaremos al aula antes que él porque se habrá detenido a hablar con un alumno, así que no tendremos ningún problema para hacerlo.

La segunda comenzará en la primera planta, concretamente en el aula 5. Allí se nos informará de que la revisión de un examen que teníamos pendiente, nos coincide con una clase. Aprovechando que, en ese preciso momento, el profesor se encuentra en la facultad y en horario de tutorías, iremos a su despacho para ver si nos la puede aplazar. Al salir del aula, nos encontraremos a una compañera en el pasillo que también quería hablar con él y nos dirá que se encuentra en la secretaría. Así que iremos hacia allí pero cuando lleguemos, nos dirá que está muy ocupado para atendernos.

Ya decididas las historias se procederá a editar los archivos `XML`. Vamos a hacer referencia a los protagonistas que aparecerán en un momento determinado como puede ser Miguel, aquí se verá como queda uno, después de añadirlo.

```
<PersonajePrincipal id="pp1" idHistoria="Historia1" idObjetivo="Punto7"
    nombre="Miguel" camara="si" modelo="Prefap_ChicoDePie1"
    animado="si">
     <Posicion>
         <X>16.772</X> <Y>0.138</Y> <Z>11.038</Z>
     </Posicion>
     <Rotacion>
         <X>0</X> <Y>335.4413</Y> <Z>0</Z>
```

```
            </Rotacion>
            <Hablar>¿Dinero? Lo siento, tío. He traido lo justo para mi. No tengo
                más.</Hablar>
            <ModelizarFigura>
                <RopaSuperior>
                    <R>255</R> <G>164</G> <B>32</B>
                </RopaSuperior>
                <RopaInferior>
                    <R>30</R> <G>30</G> <B>30</B>
                </RopaInferior>
                <Calzado>
                    <R>0</R> <G>0</G> <B>0</B>
                </Calzado>
            </ModelizarFigura>
        </PersonajePrincipal>
```

Además, según los hitos propuestos, en el primero, nada más comenzar y debido al examen, el protagonista verá a más personajes cerca del aula 3, así que también tenemos que tenerlo en cuenta. Por tanto, se procederá a crear las historias editando los archivos historia.xml, objetivos.xml, protagonistas.xml y personajes.xml. Seguidamente, se muestra como sería introducir uno de los personajes en el archivo XML correspondiente.

```
<PersonajeExtra id="pe81" planta="0" modelo="Prefap_ChicoBarra1"
    animado="si">
      <Posicion>
              <X>11.785</X> <Y>0.093</Y> <Z>−3.224</Z>
      </Posicion>
      <Rotacion>
              <x>0</x> <Y>220.57</Y> <Z>0</Z>
       </Rotacion>
       <Modelizar>
              <RopaSuperior>blanco</RopaSuperior>
              <RopaInferior>gris</RopaInferior>
              <Calzado>black</Calzado>
       </Modelizar>
</PersonajeExtra>
```

Una vez terminadas, se examinará cómo se comportan todos los elementos que intervienen para su desarrollo. Cuando se acceda al modo historia a través del menú principal, se cargarán todos los cubos de posición especificados en el archivo de `facultad.xml`. De esta forma conseguiremos el mapeado de la facultad y ubicar al jugador en la posición inicial que se dio al comienzo de la explicación, conociendo el lugar y la planta en el que se encuentra. Con estos datos, todos los objetos y personajes secundarios, ubicados en los archivos `objetos.xml` y `personajes.xml` respectivamente, que tengan asociada dicha planta, se cargarán en el juego.

Continuando con el desarrollo de la historia, el jugador aparecerá en la puerta de la secretaría, como se ve en la Figura 10.2. Sin embargo, no empieza

inicialmente con una predefinida porque, como se explicó en el apartado 6.4, esta estructura sigue el modelo de `Árbol Invertido`. Es decir, el usuario tiene la libertad de recorrer la facultad libremente. El narrador no comenzará a darle indicaciones a menos que entre en alguna a través de su primer objetivo. Según el enunciado, tampoco existe un orden concreto de comienzo de las historias ya que puede que al jugador le interese hablar con el profesor para cambiar la revisión antes que realizar el examen. Es decir, puede comenzar la segunda hasta terminarla, y después continuar con la primera. Para este ejemplo vamos a suponer que el usuario quiere empezar la primera, pero en mitad de ella, quiere cambiar a la segunda y acabarla. Finalmente volverá en el momento en el que dejó la anterior y la finalizará, completando así el juego.

Figura 10.2: Ubicación inicial del jugador en la historia

Figura 10.3: Primer objetivo de la primera historia

Cuando se inicie el primer objetivo de la primera historia, el narrador se activará y comenzará a relatar a partir de ese momento. Para lograr esto, se hará uso del `TTS` explicado en el apartado 8. Además, se muestran los mensajes asociados (como se puede ver en la Figura 10.3), durante el tiempo que se tarde en narrarlos (en el caso de la plataforma de *Windows* se establecerá un tiempo determinado que varía en función de la longitud de cada uno de ellos).

Siguiendo con el principio del argumento de la historia, Gabi (el protagonista) no sabe de la existencia del examen por lo que se sorprende al ver a tantos de sus compañeros en el pasillo, en frente del aula 3. Por ello, el primer *cubo de historia* puede situarse en cualquier punto del pasillo, porque los demás estudiantes pueden ser vistos desde ambos lados. Sin embargo, como se le sugiere que se aproxime para averiguar lo que sucede, el primer objetivo tendrá que posicionarse en un sitio más alejado de ellos, de forma que el mensaje tenga sentido cuando el narrador le proponga que se acerque. Así que se colocará en el norte del pasillo oeste de la primera planta ya que el aula 3 está muy próxima al acceso que da a las escaleras del sur. Además, se le asignará un tamaño lo suficientemente grande como para que no se pueda cruzar el pasillo desde el norte, sin haber atravesado el cubo de historia anterior. La dimensión y la posición finales se pueden ver en la Figura 10.4, así como los alumnos a la espera del examen al fondo.

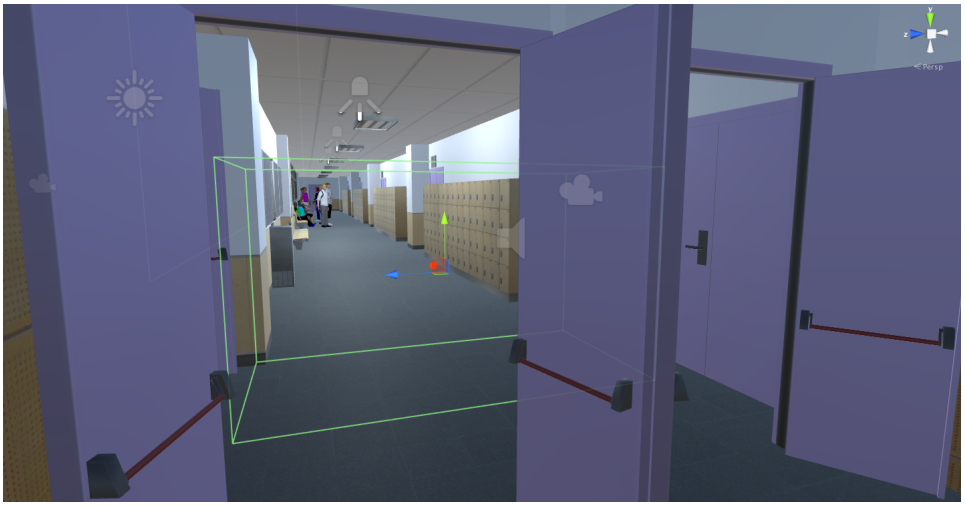


Figura 10.4: Dimensión y posición del primer cubo de historia

En el segundo objetivo, el *cubo de historia* es más extenso como se aprecia en la Figura 10.5. Esto es debido a que puede ocurrir que el jugador, si lo considera necesario, se acerque a sus compañeros por la escalera sur.

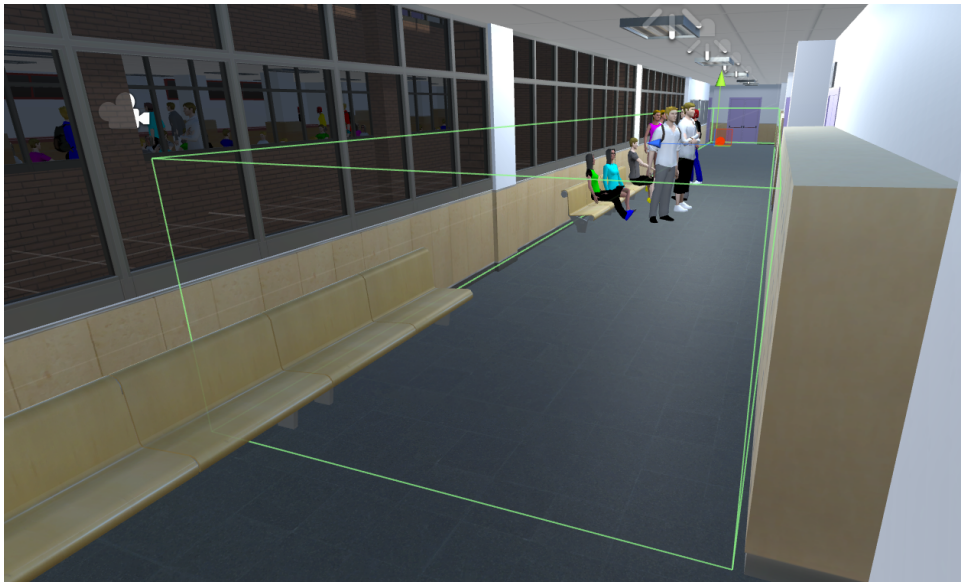


Figura 10.5: Dimensión y posición del segundo cubo de historia

Gabi en ese momento se da cuenta de lo que realmente pasaba (segundo *cubo de historia*). Tras esto, el narrador le sugiere ir a la máquina expendedora a comprar un refresco para calmar los nervios (tercer *cubo de historia*). Inicialmente se puede pensar en ambos mensajes como uno solo, constituyendo un único objetivo, ya que en el primero no se le indica que vaya a un lugar en concreto. Sin embargo, éste puede resultar demasiado abultado para que el jugador pueda seguir jugando (el texto que se muestra ocuparía más de la mitad de la pantalla). Por ello, se decide dividirlo en dos *cubos de historia*, situados en la misma posición y con las mismas dimensiones. Esto es posible gracias a la acumulación de los mensajes de los objetivos conforme el jugador avanza en la historia (explicada en el apartado 8.4) que permite que se muestren y se relaten en orden, sin que se superpongan entre ellos.

Cuando Gabi llegue a la máquina expendedora, el narrador nos avisará de que no disponemos de dinero suficiente para comprar un refresco. A continuación, nos enviará a la cafetería en busca de alguien que nos lo preste. En este caso, volvemos a tener el mismo dilema que con los dos cubos anteriores, ya que el mensaje resultante también es demasiado extenso. Así que, adoptando la misma medida, se usarán dos *cubos de historia* (cuarto y quinto objetivo). En la Figura 10.6. se puede ver el resultado final de ambos.

(a) Cuarto mensaje de historia

(b) Quinto mensaje de historia

Figura 10.6: División de los mensajes en dos objetivos diferentes

En este punto de la historia, ignoraremos las indicaciones del narrador para ver qué sucede si le llevamos la contraria. En lugar de dirigirnos a la cafetería, daremos media vuelta y subiremos las escaleras hasta el aula 5 de la primera planta. Es decir, desde el pasillo este en el que se encuentra la máquina, accederemos a las escaleras del norte. Cuando crucemos la entrada y atravesemos el cubo de posición que la delimita (en el apartado 5.4 se explica el método de localización del usuario que emplea estos tipos de cubos), el algoritmo de búsqueda del camino mínimo hasta el objetivo (en este caso, la cafetería) detectará que nos estamos desviando de él. Es entonces cuando entrará en juego el narrador, el cual nos dará a conocer esta situación mediante mensajes de control que se describieron en el apartado 8. Además, la forma en la que se dirija al jugador variará dependiendo del número de lugares que éste

atraviese erróneamente, tomando una actitud más desagradable si persiste en ello.

Por lo tanto, el narrador le hará saber que se está alejando mediante mensajes que se generarán a partir de frases definidas en `mensajesControl.xml`. Este desvío constituye un fallo, por lo que el narrador comenzará por un nivel de enfado bajo. Sin embargo, si el jugador persiste en ello, aumentará a grados más elevados, como veremos más adelante. El texto que se le muestra se puede ver en la Figura 10.7 y las oraciones del archivo que lo forman en son las siguientes:

```
<ContenedorFrases>
    <Niveles>
    <Nivel id="1">
        <Frases>
            <Frase sitio="no">
                <Mensaje>Teniendo consciencia de que este no era el camino a seguir,
                    Gabi decidió ignorarlo</Mensaje>
            </Frase>
         </Frases>
        <Cierres>
            <Cierre minus="si">
                <Mensaje>sin saber las consecuencias que provocaría</Mensaje>
            </Cierre>
        </Cierres>
    </Nivel>
    </Niveles>
</ContenedorFrases>
```

Figura 10.7: Mensaje del narrador cuando el jugador se desvia

Cuando se encuentre en la primera planta, el sistema detectará un vez más el desvío. El narrador procederá a volver a indicárselo, pero esta vez cuando salga de las escaleras a alguno de los dos pasillos. Esto es debido a que no se lanza un mensaje cada vez que atraviesa un lugar incorrecto en la ruta porque, en ese caso, el narrador le estaría sugiriendo constantemente por donde tiene ir y lo que se pretende es darle más libertad para que lo decida por el mismo. Si se empeña en ignorarlo y, por ejemplo, entrar a varias aulas, entonces el narrador alcanzará el último nivel. No volverá a comunicarse con él hasta pasado un tiempo en el que le recordará el lugar al que tiene que dirigirse para completar la historia que esté en curso. Otro evento importante que sucede en este punto es la destrucción de los objetos y personajes de la antigua planta, y la instanciación de los pertenecientes a la nueva.

A continuación se entrará en el aula 5, dentro de la cual se encuentra el primer objetivo de la segunda historia. Cuando se acceda a él, el hilo de la primera historia se interrumpirá y la que se esté desarrollando pasará a ser la segunda. Sin embargo, se conservará el objetivo por el que se había quedado el usuario por si decide retomarla. A partir de ese momento, el sistema cargará los siguientes de la nueva historia, a medida que se vayan completando, y actualizará las nuevas rutas hacia estos. El narrador, por su parte, informará de los nuevos caminos y el nivel de enfado que tenga en ese momento se reiniciará como si del principio del juego se tratara. Todos estos procesos son posibles gracias a la estructura de `Árbol Invertido` antes mencionada.

Tras salir del aula 5 para ir al despacho 450, Gabi se encuentra a Macarena (tercer objetivo) que le informa que el profesor se encuentra en la cafetería (cuarto objetivo). Por ello, en este punto, entrarán en juego los protagonistas de las historias. Para construir este personaje, se tendrá que editar el archivo `protagonistas.xml`. Lo más destacable de ellos y que los diferencia de los de relleno en los archivos `XML`, aparte del mensaje de historia que contienen, son los puntos a los que se moverán.

En el cuarto objetivo, Macarena se comunicará con Gabi. Éste debe acercarse a ella a una cierta distancia. Tras ello, la protagonista se girará mirándole de frente y entonces saldrá en la pantalla el texto que tenga asociado. Además, se desactivarán los controles del jugador, con el objetivo de que esté atento cuando se le muestre el mensaje como en la Figura 10.8. Al acabar la conversación y antes de salir del aula, Gabi encontrará el quinto objetivo en el que el narrador le sugerirá ir a la secretaría

Tras ello, Gabi bajará las escaleras hasta la secretaría donde encontrará a Don Luis, el cual le dirá que está muy ocupado para atenderle. En este quinto objetivo, también se usará un protagonista animado, pero a diferencia del anterior sin movimiento. En la Figura 10.9. se puede ver el mensaje completo del profesor Don Luis.

Figura 10.8: Diálogo con la protagonista

Figura 10.9: Diálogo con el profesor Don Luis

Finalmente, acabaremos la historia al terminar de hablar con el profesor y al acercarnos a la salida de la cafetería. El narrador la terminará haciendo una reflexión sobre lo ocurrido. Con ello, concluirá la segunda y el jugador entrará automáticamente en la primera en el mismo punto en que la había dejado. La ruta hasta el siguiente objetivo se actualizará y el narrador le recordará el lugar al que tendrá que acudir para continuar el juego, como se muestra en la Figura 10.10.

Figura 10.10: Mensaje de redirección a la siguiente historia

Rememorando donde dejamos la primera historia, lo último que habíamos hecho era ir a la máquina expendedora. El último mensaje que envió el narrador fue ir a la cafetería, con lo que le haremos caso para ver lo que sucede.

Nada más entrar, accederemos al sexto y séptimo objetivo (por estar en la misma posición) en los que se nos informará que debemos pedirle dinero a un amigo de la facultad que nos debe un favor.

Figura 10.11: Descripción del personaje

Después, se realizará un cambio de cámara para centrar la atención del jugador en el amigo. Se desactivarán los controles del usuario únicamente hasta que se termine de contar la descripción del personaje que tendremos que buscar (ver Figura 10.11). Esto es debido a que éste no tiene movimiento asociado. Luego se volverá a la cámara de primera persona del jugador y se podrá continuar con el juego. Al llegar al personaje, éste nos dirá que no nos puede dejar ni una moneda. Posteriormente, en la misma cafetería nos encontraremos el octavo objetivo, en el cual el narrador hará un comentario sarcástico sobre el amigo y nos recordará que tenemos que realizar el examen.

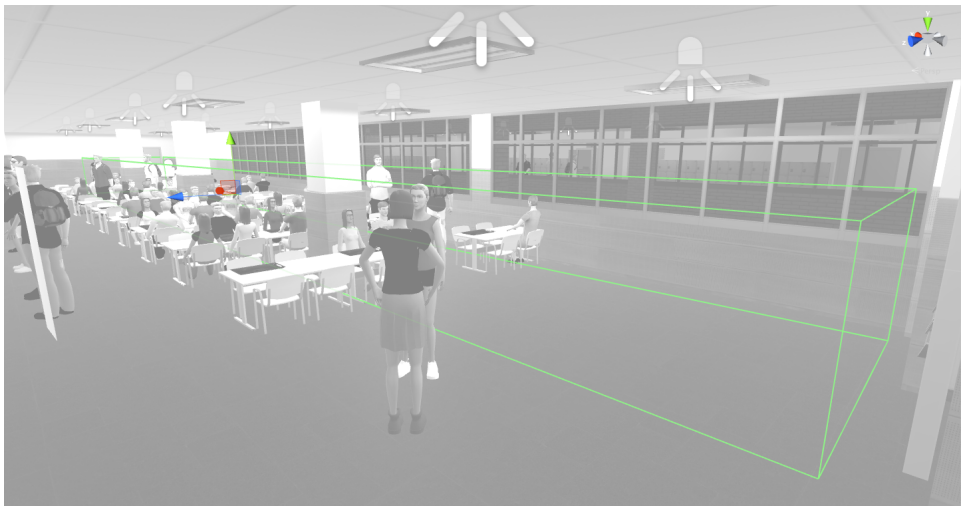


Figura 10.12: Cubo de historia que abarca dos salidas

El *cubo de historia* del noveno objetivo es ligeramente diferente al resto cubre las dos salidas de un lugar como se aprecia en la Figura 10.12. Lo que pretendemos con este cubo en la cafetería es controlar las salidas que pueda tomar el usuario para dirigirse al examen. El mensaje de este punto pretende que el jugador se de prisa en llegar al aula 3 porque el profesor está a punto de llegar. En este caso, se le asignarán los diferentes lugares por los que ha de pasar y se producirá un cambio de cámara para ver el movimiento, como se puede ver en la Figura 10.13.

Una vez dentro del aula activaremos el décimo objetivo. Éste consiste en un mensaje de preparación previa para realizar el examen, en forma de minijuego. Tomaremos asiento y procederemos a realizar el examen. En ese momento, cambiaremos de escena a la del minijuego descrito en el apartado 5.6. Cuando se finalice el test, el jugador volverá a la escena del modo historia en en mismo lugar y punto por el que había dejado la historia.

Figura 10.13: Movimiento del profesor al caminar

Al salir del aula encontraremos el último *cubo de historia*, en el que el narrador contará el mensaje que fianalizará la historia haciendo enfásis en los días que le faltaban a Gabi para terminar el curso. Tras ello, el sistema detectará que no hay más relatos disponibles a los que redirigirnos y se producirá el cambio de escena donde terminará el juego. Esta consta de un menú con el título del juego y un botón que permite salir de la aplicación. Este último `Canvas` se puede ver en la Figura 10.14.

Figura 10.14: Última escena del juego

10.3.2. Modo Editor

Es un nuevo modo que permite la integración del usuario con el desarrollo de una historia. Se ha hecho con el propósito de generar una herramienta que le permita interactuar con un objeto y a la vez con la aplicación. Así se consigue que pueda modificar los archivos `XML` para introducir cualquier tipo de elemento, sin la necesidad de disponer de la ventana `Scene`[2] que provee `Unity`. Este modo, a su vez está dividido en dos:

- `Modo edición usuario`
- `Modo de edición de objetos`

El primero de ellos es más genérico y más inmediato a la hora de conseguir la posición y la rotación, mientras que el otro está más centrado en funcionar como un programa de diseño gráfico.

10.3.2.1. Modo de edición usuario

Es aquel que se ejecuta por defecto, una vez que en la pantalla de inicio se selecciona *Modo Editor*. Está formado por un `Canvas` que muestra por pantalla la posición y la rotación que ocupamos dentro de la facultad. Cada una de estas magnitudes está definida por las variables X, Y, Z que conforman el vector de coordenadas, el cual es mostrado por los campos de texto del `Canvas` (como se puede ver en la Figura 10.15). El control del personaje, al igual que en el *Modo Historia*, se lleva a cabo mediante el `FPSController` que conforme lo vamos desplazando va actualizando los campos.

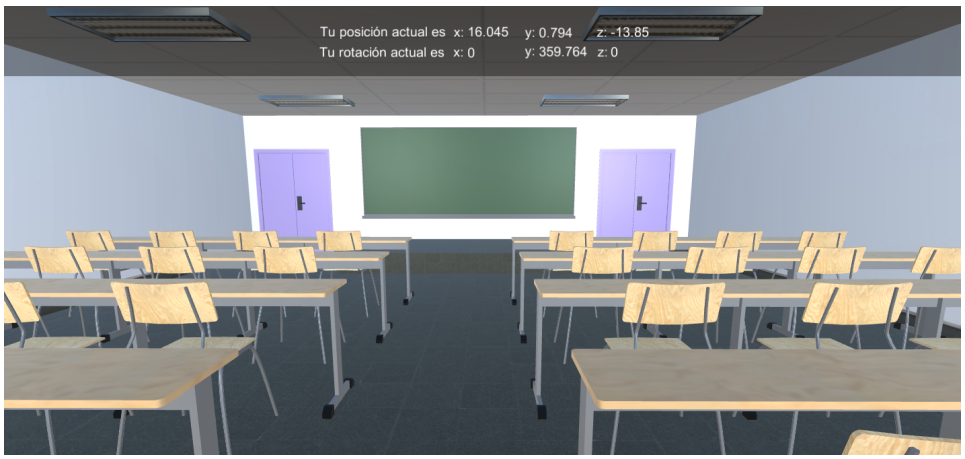


Figura 10.15: Modo edición usuario

[2]Es una ventana que permite introducir y modelar objetos desde `Unity`.

Se pensó sobretodo en ofrecer una manera muy sencilla al usuario de conseguir los valores deseados de forma rápida, para poder introducir un objeto en los archivos `XML` sin tener que detenerse demasiado. El principal inconveniente es lo poco preciso que es si se quiere introducir un elemento pequeño como por ejemplo un libro. Para solucionarlo creamos el *Modo de edición de objetos*.

10.3.2.2. Modo de edición de objetos

Proporciona al usuario, mediante un menú, la posibilidad de moldear, alterar y modificar un objeto `Cube` del juego como quiera. La activación de éste se produce cuando se pulsa la tecla E desde el *modo de edición de usuario*, habilitando un Canvas dividido en tres apartados: posición, rotación y escala. Por otro lado el objeto que aparecerá siempre dentro del campo de visión del jugador (como se puede ver en la Figura 10.16) y tres botones relativos al tipo de objeto que se desea editar. Es importante resaltar que no pueden estar activos los dos al mismo tiempo, por lo tanto mientras uno esté en ejecución el otro permanecerá inactivo.

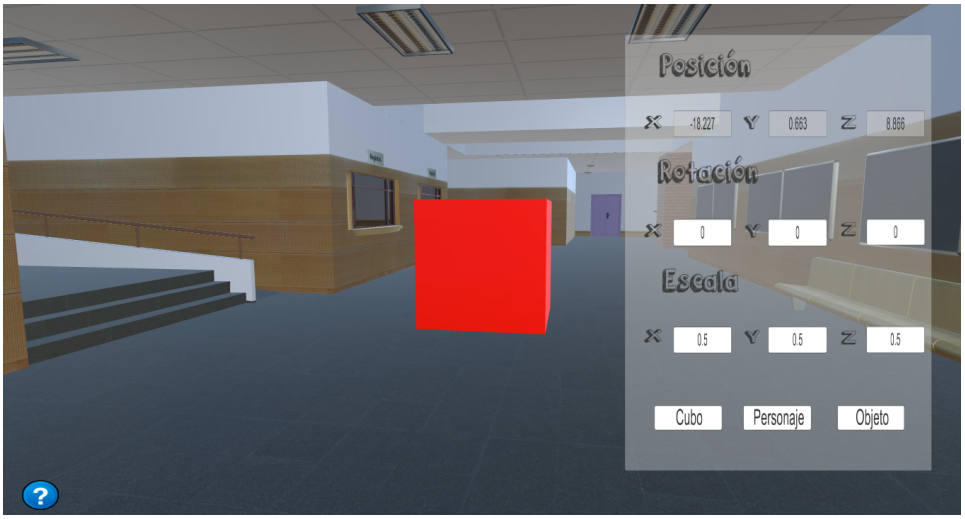


Figura 10.16: Modo de edición de objetos

El objeto, el cual manejará el usuario, contiene dos `scripts`. Uno sigue en todo momento al jugador, haciendo que esté siempre visible desde la cámara del `FPSController`. Para ello, necesitamos la función `ScreenToWorldPoint` que tiene la cámara, la cual transforma la posición del objeto con el fin de verlo en el espacio de pantalla. El otro le da el control al usuario para que pueda moverlo sobre cualquiera de los tres ejes X, Y, Z. Este movimiento lo

podrá llevar a cabo mediante las teclas W,S (desplazamiento hacia arriba y hacia abajo) relacionado con el eje Y, las teclas A,D (desplazamiento horizontal y vertical) correspondiente con el eje X y las teclas Flecha Arriba y Flecha Abajo (desplazamiento hacia adelante y hacia atrás) correspondiente al eje Z. Los tres componentes del vector se verán reflejados en el campo de posición del `Canvas` y se irán actualizando conforme el usuario vaya desplazando el objeto.

Tanto la rotación como la escala, se encuentran definidos por el componente `InputField`, perteneciente a los elementos `UI`. Es un campo que se edita pulsando con el ratón y que permite limitar el tipo de contenido, en nuestro caso para que sólo deje introducir dígitos. De esta forma se consigue validar los datos impidiendo al usuario poner letras. Ambos tienen asociada una función que se realizará tras modificar alguno de estos campos, se encargará de llevar los cambios pertinentes que hayamos efectuado al objeto en tiempo de ejecución.

La primera vez que se pasa del *modo de edición de usuario* a éste, la posición obtiene como valor predeterminado la que nosotros le hemos asignado dentro de `Unity`, es decir, la correspondiente a la puerta de entrada principal de la facultad. Como más tarde veremos, esta posición inicial depende de otros factores. La rotación del objeto inicial adquiere el valor 0,0,0 para las tres componentes y la escala, por defecto, 0.5,0.5,0.5. Si por el contrario, se ha cambiado de modo más de una vez, adquirirá los últimas coordenadas dadas.

La finalidad de crear este modo viene dada por la necesidad de solventar el contratiempo del anterior, permitiendo de esta forma dar un control de edición mayor. La razón es que se ajusta más a la necesidad del usuario en el momento en el que quiere añadir un elemento con cualquier tipo de dimensión, como puede ser una bandeja, un plato, una bebida, un personaje, etc.

Por este motivo incorporamos tres botones al `Canvas`, permitiendo manejar tres tipos de objetos, cada uno de ellos pensado para una finalidad. El *botón Cubo* carga un cubo que se relacionará en el modo historia con los cubos de posición y de historia, el *botón Personaje*, servirá tanto para añadir personajes como protagonistas y el *botón Objeto*, que en este caso es un libro. Todos ellos tienen el fin de ayudar al usuario a que introduzca de forma sencilla los elementos que comprenden las historias.

A continuación se presenta un ejemplo donde se aplicará esta herramienta explicando todos los pasos a seguir, para introducir a un personaje dentro del aula 2 con un libro sobre la primera mesa.

En primer lugar, desde la pantalla de inicio debemos acceder al Modo Editor. Una vez en él, iremos al aula 2 donde presionaremos la tecla E para abrir el *modo de edición de objetos*. Se nos mostrará en frente de nosotros el cubo,

que se inicializa por defecto, a continuación pulsaremos el *botón de Personaje* para moldearlo y colocarlo en la posición deseada, tras realizar este paso modificaremos las coordenadas de posición, rotación y escala hasta que nos quede el personaje como en la Figura 10.17. Una vez realizado, pasaremos a introducir el libro. Para ello pulsaremos el *botón de Objeto* del Canvas hasta situarlo sobre la primera mesa como se ve en la Figura 10.18. Ahora ya podemos añadir los datos al XML de personajes y objetos para instanciarlos en el modo historia como podemos ver en los fragmentos que se muestran a continuación, llegando así al final del proceso. El resultado lo podemos apreciar en la Figura 10.19.

```
<PersonajeExtra id="pe83" planta="0" modelo="Prefap_ChicoDePie1"
    animado="no">
    <Posicion>
        <X>7.06</X>
        <Y>0.144</Y>
        <Z>−9.552</Z>
    </Posicion>
    <Rotacion>
        <x>0</x>
        <Y>40</Y>
        <Z>0</Z>
    </Rotacion>
    <Modelizar>
        <RopaSuperior>azul</RopaSuperior>
        <RopaInferior>blanco</RopaInferior>
        <Calzado>negro</Calzado>
    </Modelizar>
</PersonajeExtra>
```

```
<Objeto id="56" planta="0" modelo ="Prefap_Libro1">
        <Posicion>
            <X>7.719</X>
            <Y>0.822</Y>
            <Z>−8.997</Z>
        </Posicion>
        <Rotacion>
            <X>0</X>
            <Y>90</Y>
            <Z>0</Z>
        </Rotacion>
 </Objeto>
```

A veces, se puede dar el caso de que moleste el `Canvas` mientras se intenta situar un objeto. Con la tecla H conseguiremos que se oculte, desapareciendo de la pantalla pero manteniéndose activo, y para volver a mostrarlo se pulsará la tecla J. Si se desea salir de este modo y volver al modo de edición de usuario bastará con pulsar la tecla Q.

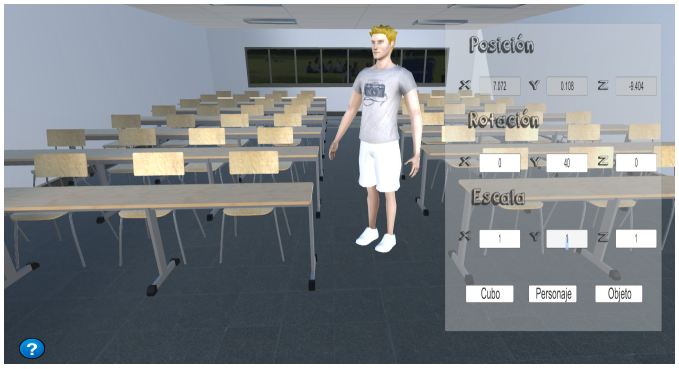


Figura 10.17: Personaje tras colocarlo en el modo de edición de objetos

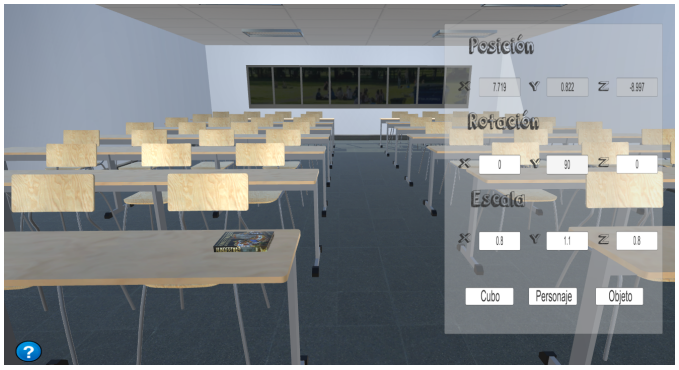


Figura 10.18: Libro tras colocarlo en el modo de edición de objetos

Figura 10.19: Resultado en modo historia, tras introducir el personaje y el libro.

10.3.2.3. Ayuda al usuario

La cantidad de comandos que el usuario ha de conocer para utilizar el *modo de edición de objetos* sin las indicaciones apropiadas resulta muy complicado. Al estar inhabilitado y por tanto oculto, el jugador desconoce su existencia y su funcionamiento. Por este motivo se decidió implementar un botón de ayuda (como se puede ver en la Figura 10.20) que mostrase la información necesaria para dar a conocer todas las opciones que ofrece este modo a través de un `Canvas`. Permanecerá siempre disponible mientras que se encuentre en él.

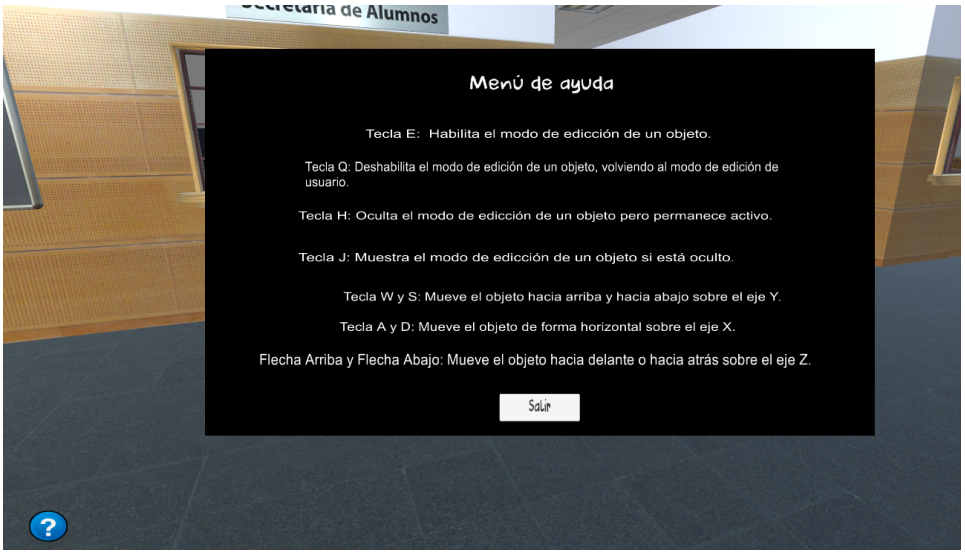


Figura 10.20: Ayuda

Con el propósito de darle una mayor visibilidad cuando se pulse, se habilitará dicho `Canvas` y se deshabilitará cualquier otro que esté en la escena incluso cuando esté en el modo de edición de usuario.

10.3.3. Característica común de ambos modos

La manera de interactuar entre los dos modos no era posible, ya que cuando entrabas en uno no había ningún sistema que te permitiese cambiar al otro. Nos pareció interesante que si un usuario estaba creando una historia, tuviese un mecanismo capaz de pasar al *Modo Edición* y conservase la posición actual del *Modo Historia*. Así, en el caso de que quisiese introducir un objeto o personaje y necesitase las coordenadas para el `Xml` correspondiente, le evitábamos el hecho de tener que salir de la aplicación, volver a iniciarla y

entrar en el modo de edición para situarse por donde estaba anteriormente en la historia y obtener los datos.

Por esta razón se implementó un menú para salir del modo actual (como se puede ver en la Figura 10.21) generado por un `Canvas`, el cual consta de un texto que nos reitera si realmente deseamos salir de ese modo junto con dos botones uno de confirmación (Sí) y otro de cancelación (No). Para acceder a él, se ha de pulsar la tecla *Esc*. Si se clickea sobre el botón de afirmación nos llevará a la *pantalla de inicio*, desde la que se puede elegir cualquiera de los dos modos. En caso contrario permanecerá en el que nos encontremos. Este menú está accesible en ambos con una pequeña limitación en el *Modo Historia*.

Figura 10.21: Menú para salir del modo actual

Esa limitación reside en que el usuario tendrá deshabilitado el menú mientras que se esté produciendo un cambio de cámara o esté hablando con algún personaje y por tanto, aunque presione *Esc*, no se le mostrará. Se decidió poner estas condiciones porque son momentos dentro de la historia que pueden resultar claves para el desarrollo de la misma y se quiere que el jugador preste atención.

Capítulo 11

Trabajo individual

Ninguno de nosotros es tan bueno como
todos nosotros juntos.
Ray Kroc

Desde el principio, los dos integrantes estábamos interesados en participar en todas las partes que conforman el proyecto. Por esta razón, toda el desarrollo lo llevamos a cabo de forma conjunta, trabajando en casi todo momento dentro de la facultad de informática o bien desde casa a través de aplicaciones de comunicación en tiempo real.

11.1. Frank Vito Julca Bedón

Para empezar y tras escoger este proyecto de fin de grado de manera conjunta, investigué sobre el motor de videojuegos sobre el que se desarrollaba, haciendo las primeras pruebas y creando los primeros proyectos.

Cuando nuestros tutores nos hicieron entrega del modelo de la facultad en formato `Maya` para utilizarlo como entorno gráfico de nuestra aplicación, decidimos el sistema operativo en que se desarrollaría el proyecto ($OS\ X$) y entre los dos nos pusimos manos a la obra con todo lo que habíamos aprendido cada uno. Aunque posteriormente fuimos aprendiendo más con el paso del tiempo, decidimos comenzar cuanto antes, por todos los hitos iniciales que nos habíamos propuesto.

Así que utilizando el modelado del edificio y el entorno `Unity` comenzamos el proyecto. El primer paso fue dotarlo de la física necesaria para que el jugador pueda interactuar con él y sin caerse al vacío como sucedía en un principio. A continuación, nos dimos que cuenta que, pese a esto último, el usuario no

era capaz de desplazarse correctamente dentro facultad. Para solucionarlo, optamos por crear un mecanismo similar a como funcionan las puertas de un edificio. Utilizando un objeto `Cube` que actúa como bisagra y un otro que activa el sistema, se consiguió el resultado deseado. Obtener las posiciones de los cubos y generar el archivo `XML` resultó tedioso, así que nos dividimos este trabajo por plantas. Yo me encargué de realizar los de la planta baja.

Entre las múltiples posibilidades de localización del jugador en el edificio, ambos escogimos e implementamos el *método por duplicidad*. Al igual que sucedía para el mecanismo anterior, había demasiados cubos que situar (dos por cada puerta). Así que decidimos repartirnos el trabajo de nuevo. En este caso, yo me encargué de realizar los de la primera planta.

Desde un principio, tanto mi compañero como yo, queríamos realizar un minijuego con el que el usuario pudiera interactuar y que aportara algo diferente en las historias. Por este motivo, decidimos implementarlo por igual y aprovechar más el modelado de la facultad, puesto que se desarrolla en frente de la pizarra de un aula.

Durante el desarrollo de las historias, nos vimos en la necesidad de dividir los cubos en dos tipos: cubos de posición y de historia. Los *cubos de posición* se utilizan para localizar al usuario y los *cubos de historia* para el desarrollo del juego. Los cubos de posición seguían siendo los mismos que antes colocamos, pero la posición y dimensión de los nuevos cubos de historia eran otras. Como es el usuario el encargado de dar estas coordenadas para situarlos, no hizo falta hacer división del trabajo para lograr este cometido. Para relacionar los cubos de historia entre ellos, se pensaron en varias estructuras que hicieran posible. Al final, elegimos la de tipo `Árbol Invertido`, el cual no tiene prefijadas historias con un único inicio y final.

Una vez resueltas la relaciones entre las historias mediante dicha estructura, consideramos utilizar algoritmos de búsqueda de camino mínimo para obtener la ruta hasta el siguiente objetivo. De entre las múltiples posibilidades nos encontramos con `Dijkstra`, la cual fue una buena opción, debido al coste que supone dicho algoritmo. Aunque, de entre los que analizamos, `Dijkstra` ofrece una facilidad para su implementación que los demás no tienen. Por ello, decimos integrarlo en el proyecto adaptando la *matriz de adyacencia* que utiliza para que sea compatible con los cubos de posición que emplea el método de duplicidad. Esto permite que no sólo la localización del usuario sea genérica para cualquier tipo de edificio, sino también el algoritmo que se aplica sobre ella. Por último, se ha de mencionar que lo evaluamos, verificando su correcto funcionamiento con casos de prueba.

A continuación, pensamos en darle voz a las mensajes de las historias que cuenta el narrador mediante el uso de un sistema `Text To Speech`. No queríamos utilizar software externo que el usuario tuviera que descargar y apren-

der a instalarlo. Fue entonces cuando nuestros tutores nos dieron una alternativa mejor: usar el comando *Say* de la plataforma *OS X*. Eso implicó documentarnos sobre cómo gestionar los procesos del sistema desde `Unity` y terminamos adoptando esta solución. Sin embargo, este método no era compatible con otras plataformas. En el caso de Windows, decidimos usar directivas de preprocesamiento para ejecutar un mecanismo que omite la voz del narrador, pero que continúa mostrando los mensajes de las historias durante un tiempo que varía en función de su longitud.

Cuando la mayoría de los objetivos estaban cumplidos, decidimos integrar objetos y personajes para conseguir un ambiente más realista de la facultad. Entre los dos encontrados diferentes librerías y paquetes de `Unity` que lo hicieron posible, aunque también nuestros tutores nos proporcionaron algunos de ellos. En el caso de los personajes, los dividimos en dos grupos: uno de ellos que participe activamente en las historias y otro que sirva simplemente para decorar el edificio.

Posteriormente, decidimos agregar movimiento y animación en los personajes del primer grupo. Además, gestionamos los cambios de cámara oportunos focalizando la atención del usuario en ello cuando éstos estuvieran en movimiento o mientras que el narrador no dejara de relatar el mensaje de historia. También construimos varios tipos de modelo para cada grupo, dando la oportunidad al usuario de cambiar algunos detalles de ellos. En el caso del primer grupo, convenía que el usuario fuera capaz de editar más detalladamente las características de estos personajes, como por ejemplo el color de la ropa o calzado. Así que se utilizó el modelo de color *RGB*, proporcionando así una gama de colores más alta. Por otro lado, en el segundo se utilizaron los colores primarios para lograr este efecto.

El número de personajes resultante de ambos grupos era elevado y obtener la posición, rotación, color de la ropa y demás atributos no podía quitar un tiempo que podíamos emplear en avanzar en otra partes. Así que mientras continuamos con el proyecto me dediqué a añadirlos poco a poco.

Cuando estábamos finalizando el proyecto, decidimos reflejar y diferenciar claramente nuestra idea de generación de historias en tiempo real y la estructura que permite crear al usuario las suyas propias. Así que esta forma decidimos crear dos modos de ejecución: el *modo historia* y el *modo editor*. En este punto, nos encontrábamos a falta de pocas semanas para entregar el proyecto, así que decidimos dividirnos el trabajo un poco más que en ocasiones anteriores. Aunque cabe destacar que ambos colaboramos activamente en todo el proceso de implementación de ambos modos, mi compañero se encargó del modo editor y yo me encargué del modo historia.

En el momento en el que se decidió crear el *modo historia*, la mayor parte de los conceptos que habíamos implementado lo constituían. Creé varios

tipos de historias (entre las que se encuentran las de este documento) para probar dicho sistema y llevarlo al límite. Tras realizar dichas las pruebas observé varios fallos que gran parte de ellos se encontraban en la plataforma *Windows*. La última modificación en que se hizo este sistema fue para los mensajes de historia pero no para la carga de los personajes, objetos y los nuevos mensajes de control del narrador. Además tampoco se distinguían entre los tipos de personajes, por lo que los mensajes de los protagonistas tampoco se mostraban. Solucioné estos errores adaptando lo que llevábamos hecho para *OS X* a este sistema operativo y realizando la mismas pruebas que se hicieron para la primera plataforma.

Mi trabajo no acabó ahí porque también encontré que, durante el intercambio de escenas, se perdía el progreso del jugador cuando volvía al *modo historia* desde el menú de inicio o el *modo editor*. Para este caso, implementé un sistema permite almacenar todo aquello que resulta significativo para continuar con la historia, como por ejemplo los objetivos que le quedan pendiente, los personajes y objetos activos, la posición del jugador, etc. El último cambio destacable que realicé en esta parte fue el cambio de historia cuando se finaliza una. Las relaciones entre las historias estaban definidas pero no se consideraba el caso de terminar una y que hubiera más disponibles. Solucioné este inconveniente, principalmente, redirigiendo al usuario a un objetivo disponible de otra historia, actualizando la nueva ruta hasta él y enviando un mensaje al narrador para que informe de este cambio. Finalmente, si no hay más historias disponibles aparecerá un canvas final con el que terminará el juego y que yo mismo realicé.

Por último, hay que destacar que la realización de esta memoria se llevó a cabo en conjunto y los dos colaboramos por igual en todos los capítulos que la componen.

Como conclusión, resulta complicado diferenciar entre quién ha hecho una cosa y otra, ya que a lo largo de todo el proyecto hemos querido involucrarnos en todas las partes que lo forman, con la idea de que dos mentes piensan más que una.

11.2. Alejandro Martín Guerrero

En primer lugar, una vez que nos asignaron el proyecto, me dedique a buscar información relativa al entorno sobre el que ibamos a desarrollarlo. De esta forma, conseguí hacerme una idea de lo que íbamos a implementar. Además me sirvió para probar la aplicación y crear unos primeros proyectos de prueba.

Tras conseguir el modelo de nuestra facultad, en formato `Maya`, el cual nos

entregaron nuestros tutores, decidimos entre los dos iniciar esta aventura.

Lo primero que hicimos fue crear un nuevo proyecto, al que incorporamos el modelo del edificio. Para ello, hubo que materializarlo mediante `Colliders` permitiendo así integrar a un personaje `FPSController` sin que este se cayese del mismo. El siguiente paso era dar libertad al usuario para moverse dentro de la facultad, por lo que decidimos crear un sistema de apertura muy parecido al que se usa en la realidad, formado por dos elementos `Cube` que ejercen de bisagra y de activador. Este último será el que active el movimiento de la puerta cuando se acerque el jugador. Aunque era una labor sencilla, llevaba bastante tiempo por lo que repartimos el trabajo y cada uno habilitó una serie de puertas, encargándome yo de las de la primera planta.

Al finalizar esta tarea, pensamos los diferentes métodos con los que obtener la posición del usuario dentro del juego. Optamos por varios y al final implementamos entre los dos el *método por duplicidad*. Al igual que el anterior, supuso integrar una serie de cubos que también nos repartimos. En este caso, me encargue de los de la planta baja.

Uno de los objetivos que pensamos desde el principio era ofrecer al usuario un minijuego que resultase diferente de las historias, consiguiendo así introducir algo que aún no había aparecido, un entorno *2D*. Por esta razón decidimos desarrollarlo.

Lo siguiente que necesitábamos era el formato de las historias para establecer las relaciones entre ellas. Para ello, primero había que diferenciar entre los objetos que actualmente teníamos, dividiéndolos en cubos de posición y cubos de historia. De entre las tres posibles estructuras de historias que planteamos, escogimos `Árbol Invertido`. Esta opción permite al usuario disponer de varias historias por las que empezar desde el inicio del juego.

Después de concretar la estructura a seguir, se planteó la ruta que debería de tomar el jugador para llegar a los objetivos que se iban planteando en el juego. Para realizar este cálculo, implementamos de forma conjunta un algoritmo de recorrido mínimo, barajamos entre distintos algoritmos de búsqueda de caminos, entre los que se encontraban `A*`, `Bellman-Ford` y `Dijkstra`, siendo este último el que escogimos. Precisa de una *matriz de adyacencia* que indique las relaciones que existen entre los *cubos de posición*. Esta matriz la hicimos de tal forma que fuese genérica para cualquier tipo de edificio. Además se efectuó una evaluación del algoritmo para comprobar su correcto funcionamiento mediante un ejemplo.

Otra de las ideas que desde el principio tuvimos, consistía en tener un narrador que fuese explicando lo que ocurría conforme se iba moviendo el usuario, como ocurre en *The Stanley Parable*. Empezamos documentándonos sobre los distintos plugins que permitían la conversión de texto a voz(`Text-To-Speech`), y los que encontramos eran de pago. No dimos con

ningún sistema, hasta que nuestros tutores nos orientaron y nos aconsejaron utilizar el comando *Say* que tiene el sistema operativo *OS X*. Fue entonces cuando empezamos a integrarlo en nuestro proyecto después de investigar sobre los procesos necesarios para llamarlo desde `Unity`, con resultados positivos hasta que lo probamos en el sistema operativo de *Windows*. Al no contar con este comando, tuvimos que adaptarlo mediante directivas de preprocesamiento, con las cuales aunque sin la funcionalidad de `TTS` logramos que funcionase y mostrase los mensajes correspondientes por pantalla.

Llegados a este punto, pensamos en añadir objetos y personajes que sirviesen para dar una sensación más realista a la facultad. Estos últimos los dividimos en dos, unos con los que el usuario pudiese interactuar y otros que ejerciesen la función descrita anteriormente. Para conseguirlo, estuvimos mirando paquetes de `Unity` que nos facilitarán la obtención de estos modelos que más tarde cargaríamos en el juego, junto con algunas librerías que nos dieron nuestros tutores. Tras obtener una serie de ellos, decidimos implementar un sistema por el cual se pudiese cambiar la ropa de los personajes, generando así la impresión de tener muchos modelos cuando en realidad contábamos con unos pocos.

A la hora de realizar la interacción con el usuario, se nos ocurrió que, análogamente a muchos juegos y películas, cuando un nuevo personaje entra en escena, hacer un cambio de cámara. Ésto nos dió la oportunidad de centrar su atención para que se dirigiese hacia los lugares que deseábamos. Por otro lado, implementamos un procedimiento para llevar a cabo los movimientos de los personajes, asignándoles una animación, de forma que pasasen por los puntos que nosotros previamente habíamos preestablecido, adoptando en cada momento la animación que le correspondía (andar o estar en reposo).

Entrando en la última fase de implementación del proyecto, faltaba crear, una vez que tuvimos los elementos necesarios, las historias a las que jugaría el usuario y el *modo editor*, el cual le permitiría crear sus propias historias sin necesidad de tener el programa `Unity`. Decidimos esta vez encargarnos cada uno de una parte, quedando poco tiempo para la entrega, dividimos el trabajo, mi compañero se encargó de hacer las historia y yo el modo editor. Aunque los dos después participamos y ayudamos en las respectivas partes que nos habían tocado.

En primer lugar, para llevarlo a cabo, hice una pantalla de inicio que se ejecuta al inicializar la aplicación, desde la que poder acceder tanto al *modo historia* como al *modo editor*. Este nuevo modo está constituido por la facultad al igual que el de historia, el cual tiene dos submodos: uno genérico y rápido con el que obtener los datos y otro más específico. Para el primero de ellos, implementé un cuadro de texto que devolvía la posición y rotación del jugador que se actualizaba conforme se iba moviendo, lo situé en la parte superior de la pantalla de manera que estuviese siempre visible. Para el se-

gundo, realicé un panel que actúa como una herramienta de edición gráfica, donde nos aparece en pantalla un objeto, que podemos mover con las teclas del teclado y cambiar de rotación y escala en tiempo de ejecución viendo los cambios que vamos haciendo sobre él. También mediante el panel permití la opción de cambiar el objeto que es un cubo, pensado para los cubos de posición y de historia que tenemos en el otro modo, por un personaje u objeto, como un libro. De esta forma, si deseamos introducir alguno de ellos, veremos lo mismo que si lo introdujéramos en el otro modo incluyendo los datos de las magnitudes en los respectivos archivos `XML`.

Además, creé un *menú de ayuda* para explicar todas las directrices necesarias para entender este modo y acceder a él. Una de la razones es que se encuentra oculto mientras está en ejecución el otro submodo, hasta que mediante una tecla lo activamos, deshabilitando el otro, lo que hace que el usuario desconozca de su existencia a no ser que se lo indiquemos. Por último, creé un menú para salir del modo actual y volver a la pantalla de inicio, permitiendo interactuar entre el modo historia y el modo editor, el cual no funcionaba en una serie de casos en el modo historia. Lo solucioné, controlando los casos en los que se producía un cambio de cámara o se hablaba con el personaje.

Por último, hay que destacar que la realización de esta memoria se llevó a cabo en conjunto y los dos colaboramos por igual en todos los capítulos que la componen.

Como conclusión, resulta complicado diferenciar en este proyecto quién ha hecho una cosa u otra, ya que a lo largo de todo el proyecto hemos querido involucrarnos en todas las partes que lo forman, con la idea de que dos mentes piensan más que una.

Capítulo 12

Conclusiones y trabajo futuro

Lo que hacemos en la vida tiene su eco
en la eternidad.
Gladiator

12.1. Conclusiones

Han pasado más o menos ocho meses desde que nos embarcamos en este proyecto. Este capítulo marca el final de este trayecto en el que al final hemos llegado a buen puerto. Después de un viaje largo y duro en el que nos hemos tenido que enfrentar a la marea, estamos orgullosos de haber recorrido y completado con éxito este mar de esfuerzo y sacrificio que sin duda ha merecido la pena.

Hemos aprendido un lenguaje que desconocíamos (`C#`) en un entorno (`Unity`) completamente nuevo para nosotros, el cual supuso un gran reto al principio y poco a poco conseguimos superarlo, hasta llegar al punto de controlarlo.

Hemos logrado cumplir todos los objetivos que nos habíamos marcado:

- La generación de historias en tiempo real se llevó a cabo analizando los diferentes sistemas de localización que permitían conocer la posición del jugador dentro del entorno gráfico. Se creó la estructura necesaria para interactuar con diferentes historias según los movimientos del usuario, haciendo uso de un algoritmo de recorrido mínimo. También se implementó un procedimiento para mostrar y narrar los diferentes mensajes evitando su superposición y un método que los generara. Por último, se introdujeron personajes y objetos, junto con los cambios de cámara oportunos, intentando ser lo más fieles posibles a la realidad.

Todos estos puntos constituyen el propósito que habíamos pensado desde el principio.

- La posibilidad de que el usuario pueda crear sus propias historias, sin tener que acceder al programa utilizado para el desarrollo de este proyecto. Este concepto se pensó mientras se realizaban los objetivos anteriores y se decidió ampliarlos con este nuevo hito. Suposo crear un nuevo modo de juego, el cual sirve de ayuda al usuario para que conozca las magnitudes necesarias para la edición de los objetos desde archivos externos que permiten la carga de las historias.

Este proyecto nos ha ayudado a ver el trabajo mayúsculo que hay detrás de los videojuegos. Además, hemos encontrado muchas posibilidades tanto para lograr su desarrollo como para continuar con él en un futuro. Esto nos ha permitido conocer una gran diversidad de juegos de aventura conversacional, permitiéndonos pensar en diferentes formas de ser originales y diferenciarnos de ellos. Ha habido muchas ideas que no se han podido llevar a cabo por falta de tiempo o por falta de conocimientos avanzados que aún no tenemos, pero estamos muy satisfechos con el trabajo realizado.

12.2. Trabajo futuro

Realizar este proyecto de una forma tan generalizada como se ha hecho propicia muchas líneas de trabajo futuro que puedan servir para mejorar lo ya existente o para añadir nuevas funcionalidades a la aplicación. Por tanto, en este capítulo describiremos algunas de las mejoras más relevantes que hemos encontrado.

La primera y una de las más importantes de ellas se centra en el vocabulario que emplea el narrador. Como se explicó en apartados anteriores, la frases que dice, cuando el jugador no sigue sus indicaciones y que están definidas por el usuario, se combinan para formar mensaje completos. Esta mejora consiste en llevar la creación de las oraciones a un nivel más elevado, de forma que se puedan generar a partir de palabras definidas por el usuario o analizar las ya existentes para extraer la información más importante del texto y formar distintas frases con ello. Sabemos que este paso es complicado porque requiere de procesos complejos como la *Generación Natural de Lenguaje* (*GNL*), pero es uno que puede conseguir un aspecto más humano e imprevisible del narrador. Además, el usuario no tendría la necesidad de formular las frases, sino que simplemente ampliaría el diccionario de palabras y el sistema se encargaría de procesar esta información.

Otra mejora que se puede llevar a cabo se trata de desarrollar más el modo de edición de las historias. Actualmente es posible obtener la distancia, dimen-

sión y demás atributos básicos para la instanciación de los objetos dentro de la escena. Sin embargo, sólo son proporcionados para que el jugador pueda editar, con estos datos, los archivos `XML` correspondientes. Esta ampliación consistiría en añadirlos desde el propio juego de forma que el usuario pueda cambiar entre los modos de edición y de historia sin tener que salir de la aplicación. Además, se propone hacer lo mismo para el narrador, ya que es el único elemento del juego que sólo se puede modificar desde el documento `XML`.

Una tercera ampliación consistiría en adquirir mayor control sobre la carga de los personajes. Instaciarlos y destruirlos únicamente cuando se cambie de planta puede resultar poco preciso. Por ello, resultaría útil implementar un sistema que, en lugar de realizar estos procesos dependiendo de la planta, lo hiciera en función de los lugares adyacentes a la posición del jugador. De esta manera, la carga en memoria será más reducida y el usuario no podrá ver los cambios que se producen a su alrededor.

Este proyecto también se puede considerar un videojuego, por lo que siempre hay algún aspecto en lo que se puede indagar aún más. En este caso, una de ellas es la interacción con los objetos, la cual no se llegó a completar por falta de tiempo, principalmente. Si se busca crear historias más complejas, entonces conseguir objetivos que requieran de objetos para completarlos sería una de los primeros propósitos que se llevarían a cabo.

Por último, a nivel de compatibilidad, se propone adaptar la aplicación para el sistema operativo *Linux*, completando así un proyecto capaz de ejecutarse correctamente en las tres plataformas más utilizadas (*Windows*, *OS X* y *Linux*). En el caso de *Windows*, cabe destacar que el `Text-To-Speech` no está implementado. Así que también se plantea desarrollar plugins y utilizar librerías propias de *Microsoft* para tal cometido.

Capítulo 13

Conclusions and future work

Nunca te das cuenta de lo que has hecho;
sólo puedes ver lo que queda por hacer.
Marie Curie

13.1. Conclusions

It's been about eight months since we embarked on this project. This chapter marks the end of this journey in which our objectives have been achieved. After a long and hard journey that we have had to face the tide, we are proud to have successfully completed it and the effort and sacrifice has certainly been worthwhile.

We have learned a language we did not know (`C#`) in a completely new environment (`Unity`) for us, which was a big challenge at first but gradually we could get over it, to the point of controlling it.

We have managed to meet all the goals we had set ourselves:

- The generation of real-time stories was carried out by analyzing the different tracking systems that allow knowing the player's position within the graphical environment. We have created a needed structure to interact with different stories according to the movements of the user, using an algorithm of minimum path. A method was also implemented to show and tell the different messages avoiding overlap and a method that could generate it. Finally, characters and objects were introduced, along with appropriate camera changes, trying to be as faithful as possible to reality. All these points were the purpose we had thought from the beginning.

- ✦ The possibility that the user can create their own stories without having to access to the program used for the development of this project. This concept was thought as the foregoing objectives were achieved and we decided then to expand them with this new milestone. It involved creating a new game mode, which would help the user to know the necessary magnitudes for editing objects from external files that allow the stories to be loaded.

This project has helped us to see the egregious work behind video games. In addition, we have found a lot of possibilities for its development and to continue with it in the future. This has allowed us to meet a wide variety of text adventure games, allowing us to think of different ways to be original and differ from them. There have been many ideas that have not been carried out due to our lack of time and also our lack of advanced knowledge, but we are anyway very satisfied with the work done.

13.2. Future work

Having done this project in such a widespread way, has led many lines of future work that may serve to improve the existing project or add new functionality to the application. Therefore, this chapter will describe some of the most important improvements that we have found.

The first and one of the most important of them focuses on the vocabulary used by the narrator. As explained in previous sections, the phrases he says, when the player does not follow his instructions and are defined by the user, are combined to form a complete message. This improvement leads to the creation of sentences to a higher level, so that they may be generated from words defined by the user or analyzing existing ones to extract the most important text information and to form different sentences with it. We know that this step is complicated because it requires complex processes such as *Natural Language Generation* (*NLG*), but it is one that can give the narrator a more humane and unpredictable aspect. In addition, the user would not need to formulate sentences, but simply expand the dictionary of words and the system would be responsible for processing this information.

Another improvement that can be accomplished is to further develop the edit mode of the stories. Currently it is possible to obtain the distance, size and other basic attributes for instantiation of objects within the scene. However, they are only provided for the player to edit, with these data, the corresponding `XML` files. This extension would be to add them from the game so that the user can switch between editing and story modes without having to exit the application itself. It is also proposed to do the same for the

narrator, since it is the only element of the game that can only be changed from the `XML` document.

A third extension would be to gain more control over the load of the characters. Instating and destroying them only when changing floor can be imprecise. Therefore, it would be useful to implement a system that, instead of performing these processes depending on the floor, does it according to the locations adjacent to the player's position. Thus, the memory load will be reduced and the user will not see the changes occurring around them.

This project can also be considered a video game, so there are always some aspects to further investigate. In this case, one of them is the interaction with objects, which was never completed for lack of time, mainly. If you are looking for creating more complex stories, then one of the first purposes to be carried out would be achieve goals that require objects to complete them.

Finally, at the level of compatibility, it is proposed to adapt the application for the Linux operating system, thus completing a project able to run properly on the three most used platforms (*Windows*, *OS X* and *Linux*). In the case of *Windows*, we should highlight that the `Text-To-Speech` is not implemented. So it is also aimed to develop some plugins and use the own *Microsoft* library for such a goal.

Apéndice A

Manual de usuario

A.1. Creación de los archivos

Esta aplicación te permite modificar muchos aspectos del juego como pueden ser personajes, historias y objetos, a partir de archivos en formato `XML`. Si no decides integrarlos en el juego, el sistema ejecutará unos por defecto, los cuales estarán disponibles para que puedas hacer uso de ellos o tomarlos como ejemplo para construirte tus propios archivos.

Su ubicación será distinta y dependerá del sistema operativo en el que lo ejecutes. Como se verá más adelante, todos ellos deben tener nombres específicos para que la aplicación funcione correctamente.

A.2. Windows

En la plataforma Windows, debes colocarlos en la carpeta *Skyline_data* que se encuentra en el mismo directorio que el ejecutable del juego. Dentro de ella, se encuentran otros archivos de configuración de la aplicación que no se deberán tocar.

A.3. OS X

En el sistema operativo OS X los archivos se encuentran dentro del propio ejecutable. Para acceder a ellos, haremos click derecho sobre la aplicación y luego sobre *Mostrar el contenido del paquete*. A continuación, nos encontraremos la carpeta *Contents*, en la cual se deberán colocar los archivos `XML`.

A.4. Edición de los archivos

A.4.1. Facultad

Este fichero constituye la base para el correcto funcionamiento de la aplicación, ya que es el encargado de definir el sistema de localización del jugador en el edificio de la facultad. No se recomienda modificarlo por varios motivos: uno es el tiempo requerido para establecer la ubicación de todos los elementos necesarios y otro la dificultad que supone entender el mecanismo para posicionarlos.

A continuación vemos un ejemplo de estructura básica del archivo `facultad.xml`, el cual añade una entrada en la cafetería que se encuentra en la planta baja.

```
<?xml version="1.0" encoding="UTF-8" ?>
<Contenedor>
    <PlantaBaja>
        <Lugar id="cafe1" tipo="estancia" orientacion="E">
            <Entradas>
                <Entrada id="eNorte" nombre="cafeteria" adyacente="pasillo">
                    <Orden>0</Orden>
                    <Posicion>
                        <X>2.557</X> <Y>0.934</Y> <Z>7.313</Z>
                    </Posicion>
                </Entrada>
            </Entradas>
        </Lugar>
    </PlantaBaja>
</Contenedor>
```

En el caso, de que desees añadir o modificar este fichero, debes tener en cuenta ciertos aspectos. Primero, se utilizarán etiquetas como *<PlantaBaja>* o *<PrimeraPlanta>*, que son las dos disponibles actualmente, haciendo referencia a la planta donde se desea situarlo.

A.4.1.1. Añadir un lugar

Para añadir un lugar dentro del archivo, deberás usar un elemento de tipo *<Lugar>*, con sus correspondientes atributos, y situarlo dentro de una planta. Los componentes que lo forman son:

- `Id`: es un identificador único para cada lugar, el cual servirá para especificar el sitio en el que se desarrolla una historia.
- `Tipo`: existen tres elementos que puedes poner en este campo: *estancia*, *pasillo* y *escalera*. El primero de ellos abarca: aulas, baños, secretaría, laboratorios y cafetería. Es importante ponerlo correctamente ya que

con él se calcula la localización del usuario. En este caso, como vamos a situar un elemento dentro de la cafetería, lo definimos como *estancia*.

- `Orientación`: designa los cuatro puntos cardinales que deberás introducir con su primera letra.

A.4.1.2. Añadir una entrada

Para añadir una entrada, al igual que un lugar, necesitarás definir una serie de componentes:

- `Id`: es un identificador único para cada elemento de un lugar.
- `Nombre`: es el nombre que daremos al lugar al que hace referencia la entrada.
- `Adyacente`: este campo indica el lugar más próximo al que da acceso.

Necesariamente ha de ser una *estancia*, un *pasillo* o una *escalera*. En el ejemplo vemos como la cafetería tiene como adyacente un pasillo, aunque no se especifica uno en concreto ya que el sistema es el que se encargará de determinar cuál de los disponibles es el más cercano.

Si un lugar es de tipo *estancia*, entonces deberás indicar el orden que ocupa entre los de su tipo. Por ejemplo, si se trata del aula 5, tendrás que añadir una etiqueta del tipo *<Orden>5 </Orden>*. En aquellos sitios que no tengan un orden específico, como por ejemplo un baño, deberás indicarlo con un 0.

Tras establecer estos parámetros ya puedes poner la posición que ocupará la entrada, introduciendo los valores de las coordenadas X,Y,Z. Para obtenerlas tienes a tu disposición dentro del juego un modo (*Modo Editor*) que te permite, de forma sencilla, conseguir esta información.

A.4.2. Historia

Las historias del juego se encontrarán en el archivo `historia.xml`. A continuación se explicarán los principales aspectos del juego que se pueden modificar con él, junto con sus elementos y atributos.

A.4.2.1. Estructura básica del archivo

```
<?xml version="1.0" encoding="UTF-8" ?>
<ContenedorHistorias>
    <PosicionInicial planta="baja" lugar="pE" puerta="sNorteSecretaria"/>
    <Historias> </Historias>
</ContenedorHistorias>
```

Dentro de este archivo, las historias que vayas a crear deberán de estar dentro del elemento que aparece como *<Historias>*. Tanto la primera línea, que hace referencia al tipo de codificación que se va a utilizar, como el elemento *ContenedorHistorias*, son estrictamente necesarios que aparezcan en el documento.

El jugador puede empezar el juego en un determinado lugar. Para conseguir esto, deberás utilizar la etiqueta *<PosicionInicial>* y especificar la planta, la ubicación y la entrada en la que aparecerá el personaje.

A.4.2.2. Añadir una historia

```
<Historia id="Historia1">
   <Lugares>
   </Lugares>
</Historia>
```

Una vez tengas la estructura básica de este archivo, el siguiente paso para empezar a crear una historia es añadir el elemento *<Historia>*, con su correspondiente atributo *id* que la identifica unívocamente. Dentro de la etiqueta anterior será necesario añadir la etiqueta *<Lugares>* para indicar los sitios por los que transcurre la historia principal.

A.4.2.3. Añadir un lugar

```
<Sector id = "aula5">
   <PuntosControl>
      <Punto id="Punto12" orden="1">
         <Mensaje>Un mensaje</Mensaje>
      </Punto>
   </PuntosControl>
</Sector>
```

Los distintos lugares en los que se desarrolle tu historia se designarán con las etiquetas *<Sector>* y su atributo *Id*. Este elemento sirve para buscar las entradas del lugar que se encuentran en el archivo `facultad.xml`, el cual se vio anteriormente. Además, se utilizarán las etiquetas *<PuntosControl>* para añadir los objetivos específicos dentro del sitio.

A.4.2.4. Añadir un objetivo

```
<Punto id="Punto12" orden="1">
   <Mensaje>Un mensaje</Mensaje>
</Punto>
```

Los objetivos de una historia se deberán indicar con etiquetas *<Punto>*, en las que se utilizarán dos atributos. Además del identificador, se deberá especificar, mediante un valor numérico, el orden en el que se tienen que completar los objetivos. Estos elementos tienen una posición concreta, pero sus coordenadas se especificarán en otro archivo `XML`, como se verá más adelante.

También es obligatorio que el objetivo tenga un mensaje asociado, pues el narrador se encargará de comunicárselo al jugador. Esto se realizará con el elemento *<Mensaje>* acompañado del texto correspondiente.

A.4.3. Objetivos

Los objetivos van ligados estrictamente al archivo de `historia.xml`. En éste se concretarán las posiciones que ocuparán los objetivos definidos dentro de las historias. El nombre del archivo asociado será `objetivos.xml`.

A.4.3.1. Estructura básica del archivo

La siguiente estructura es completamente necesaria para establecer el fichero de forma correcta.

```
<?xml version="1.0" encoding="UTF-8" ?>
<ContenedorObjetivos>
    <Objetivos>
    </Objetivos>
</ContenedorObjetivos>
```

A.4.3.2. Añadir un objetivo

```
<Objetivo id="Punto4">
   <Dimension>
      <X>2.908371</X>
      <Y>1.486802</Y>
      <Z>1.578308</Z>
   </Dimension>
   <Posicion>
      <X>-1.512</X>
      <Y>0.92</Y>
      <Z>5.599</Z>
   </Posicion>
</Objetivo>
```

Dentro de las etiquetas <*Objetivos*> se añadirán otras del tipo <*Objetivo*>, junto con su respectivo identificador. Estos elementos se relacionarán, a través del identificador, con los objetivos definidos en el archivo`historia.xml`, de forma que se les atribuya un sitio y un tamaño. La posición se especificará con la etiqueta <*Posicion*>, en la que habrán otros tres elementos que designarán las coordenadas X,Y y Z. Por otra parte, el espacio que ocupe se definirá con la etiqueta <*Dimensión*>, en la que se encontrarán las coordenadas que determinarán su tamaño.

A.4.4. Mensajes del narrador

Los mensajes que no tengan que ver con la historia principal pero sí con las decisiones erróneas que tome el jugador en ella, se encontrarán en el archivo `mensajesControl.xml`.

A.4.4.1. Estructura básica del archivo

```
<?xml version="1.0" encoding="UTF-8" ?>
<ContenedorFrases>
   <Niveles>
   </Niveles>
<ContenedorFrases>
```

Dentro del elemento <*ContenedorFrases*> se encuentra la etiqueta <*Niveles*>, la cual contiene cada uno de los niveles de intensidad con los que el narrador dirá sus frases.

A.4.4.2. Añadir un nivel

```
<Nivel id=?1?>
   <Frases>
   </Frases>
   <Aperturas>
   </Aperturas>
   <Cierres>
   </Cierres>
</Nivel>
```

Al igual que sucede con la mayoría de etiquetas, para añadir un nivel se realizará con la etiqueta correspondiente (<*Nivel*>) y su identificador. Cada uno de ellos contiene las oraciones que compondrán el mensaje, las cuales están clasificadas en tres tipos: *frase principal*, *frase de apertura* y *frase de cierre*. Todos ellos se combinarán para formar el texto completo que será narrado.

A.4.4.3. Añadir una frase principal

```
<Frase sitio="no">
   <Mensaje>Gabi decidió ir en dirección opuesta</Mensaje>
</Frase>
<Frase sitio="actual">
   <Mensaje>Gabi se dirigió</Mensaje>
</Frase>
<Frase sitio="objetivo">
   <Mensaje>Gabi recordó que su objetivo principal era ir</Mensaje>
</Frase>
```

Las frases principales son aquellas que contienen la información más importante que el narrador comunicará al jugador, en caso de que éste se desvíe del camino adecuado. Además, es el único tipo que puede constituir un mensaje por sí mismo, sin necesidad de otros para formar uno completo. Se añaden con la etiqueta *<Frase>*. A través del atributo *sitio* es posible indicar si se desea hacer referencia a un lugar específico de la facultad. En caso afirmativo, se indicará con *objetivo* o *actual*, dependiendo de si se trata del lugar del objetivo o del sitio al que acaba de acceder el jugador. Para finalizar, el texto deberá situarse en una etiqueta *<Mensaje>*, y ésta a su vez, dentro de la etiqueta *<Frase>*.

A.4.4.4. Añadir una frase de apertura

```
<Apertura minus="no">
   <Mensaje>¡Vaya!</Mensaje>
</Apertura>
<Apertura minus="si">
<Mensaje>Parece que</Mensaje>
</Apertura>
<Apertura minus="coma">
   <Mensaje>En un alarde de rebeldía</Mensaje>
</Apertura>
```

Las frases de apertura son las primeras oraciones de los mensajes. Se añaden con la etiqueta *<Apertura>* y un atributo *minus*. Éste indica si la apertura necesita un punto, una coma o ningún signo de puntuación adicional al final de la oración. Como sucede en las frases principales, se deberá colocar una etiqueta *<Mensaje>* en su interior con el correspondiente texto.

A.4.4.5. Añadir una frase de cierre

```
<Cierre minus="punto">
   <Mensaje>Ahora ya lo has entendido, ¿verdad?</Mensaje>
</Cierre>
<Cierre minus="no">
   <Mensaje>¿Seguro que quieres seguir con este despropósito?</Mensaje>
</Cierre>
```

Las frases de cierre son aquellas que terminan los mensaje. Se añaden con la etiqueta *<Cierre>* y un atributo *minus*. Éste indica si el cierre necesita de un punto al final. Esto es debido a que se pueden introducir frases interrogativas o exclamativas que no necesitan de este signo de puntuación. Como sucede con los otros tipos, se deberá colocar una etiqueta *<Mensaje>* en su interior con el correspondiente texto.

A.4.5. Minijuego

También es posible editar algunos aspectos del minijuego en la pizarra. Concretamente, las preguntas del test que aparecen en ella. Estas modificaciones se realizan sobre el archivo `preguntas.xml`.

A.4.5.1. Estructura básica del archivo

```
<?xml version="1.0" encoding="UTF-8" ?>
<ContenedorPreguntas>
   <Preguntas>
   </Preguntas>
</ContenedorPreguntas>
```

El elemento raíz del archivo es *<ContenedorPreguntas>*, el cual contiene la etiqueta *<Preguntas>* en la que se deberán añadir todas las preguntas y respuestas presentes en el minijuego.

A.4.5.2. Añadir una pregunta

```
<Pregunta id="p1">
<Enunciado>¿Cuál es la capital de Bielorrusia?</Enunciado>
         <Opciones>
          </Opciones>
</Pregunta>
```

Para añadir una pregunta, primero se deberá usar la etiqueta *<Pregunta>* con su respectivo *id*. Tras ello, se añadirá el enunciado de dicha pregunta con el elemento *<Enunciado>* y el texto en su interior. Por último, se indicarán las respuestas posibles dentro de la etiqueta *<Opciones>*.

A.4.5.3. Añadir una opción

```
<Opcion num="1" correcta="no">
  <Nombre>Sarajevo</Nombre>
</Opcion>
```

Se podrán agregar un máximo de cuatro respuestas por cada pregunta. Para ello, se deberá utilizar la etiqueta *<Opcion>* más dos atributos. El primero de ellos es *num*, el cual indica el número de respuesta y sirve a modo de identificador. El segundo es *correcta*, el cual indica si la respuesta es correcta o no. Por último, se añadirá el texto de la respuesta dentro de la etiqueta *<Nombre>*.

A.4.6. Objetos

Esta aplicación también permite añadir y modificar objetos del juego como pueden ser las bandejas de la cafetería o los portátiles de los alumnos. Estos cambios se verán reflejados en un archivo llamado `objetos.xml`.

A.4.6.1. Estructura básica del archivo

```
<?xml version="1.0" encoding="UTF-8" ?>
<ContenedorObjetos>
  <Objetos>
  </Objetos>
</ContenedorObjetos>
```

Dentro del elemento *<ContenedorObjetos>*, se situará la lista de objetos que aparecen en el juego usando la etiqueta *<Objetos>*.

A.4.6.2. Añadir un objeto

```
<Objeto id="1" planta="0" modelo="Coca-Cola">
  <Posicion>
    <X>10.38</X>
    <Y>1.18</Y>
    <Z>11.47</Z>
  </Posicion>
  <Rotacion>
    <X>0</X>
    <Y>0</Y>
    <Z>0</Z>
  </Rotacion>
</Objeto>
```

Para añadir un objeto se empleará la etiqueta *<Objeto>* y tres atributos: el identificador, el número de planta en el que se encuentra y el modelo del objeto que se desee utilizar. Los modelos de los objetos disponibles son:

- Coca-Cola
- Prefap_Monster
- Prefap_OneShot
- Prefap_Iphone
- Prefap_Mac
- Prefap_Bandeja
- Prefap_Plato
- Prefap_Cafe
- Prefap_HP

Además se deberá especificar las coordenadas de la ubicación y la rotación del objeto con las etiquetas *<Posicion>* y *<Rotacion>*, respectivamente.

A.4.7. Personajes

Los personajes que no tienen una repercusión directa en las historias se encuentran en el archivo `personajes.xml`. Es por ello que estos personajes son más conocidos como extras o de relleno.

A.4.7.1. Estructura básica del archivo

```
<?xml version='1.0' encoding='utf-8'?>
<ContenedorPersonajes>
   <PersonajesExtras>
   </PersonajesExtras>
</ContenedorPersonajes>
```

Dentro del elemento *<ContenedorPersonajes>*, se situará la lista de personajes que aparecen en el juego usando la etiqueta *<PersonajesExtras>*.

A.4.7.2. Añadir un personaje de relleno

```
<PersonajeExtra id="pe1" planta="0" modelo="Prefap_ChicaSentada"
    animado="no">
   <Posicion>
     <X>25.406</X>
```

```
        <Y>−0.225</Y>
        <Z>−9.421</Z>
    </Posicion>
    <Rotacion>
        <X>0</X>
        <Y>0</Y>
        <Z>0</Z>
    </Rotacion>
    <Modelizar>
        <RopaSuperior>rojo</RopaSuperior>
        <RopaInferior>blanco</RopaInferior>
        <Calzado>negro</Calzado>
    </Modelizar>
</PersonajeExtra>
```

Para añadir un personaje se debe usar la etiqueta *<PersonajeExtra>* y cuatro atributos: su identificador, el número de la planta en la que se quiera situar, el modelo del personaje que se desee utilizar y, por último, el atributo *animado* que indica si el personaje incorpora la animación de estar en reposo. Los modelos de personajes disponibles son:

- Prefap_ChicaSentada
- Prefap_ChicaSentada2
- Prefap_ChicaDePie1
- Prefap_ChicaDePie2
- Prefap_ChicaDePie3
- Prefap_ChicaBarra1
- Prefap_ChicoBarra1
- Prefap_ChicoDePie1
- Prefap_ChicoDePie2
- Prefap_ChicoDePie3
- Prefap_ChicoSentado
- Prefap_ChicoSentado2

Además, se deberán especificar las coordenadas de la ubicación y rotación del personaje con las etiquetas *<Posicion>* y *<Rotacion>*, respectivamente. También se permite modificar los colores de la ropa del personaje a través de los elementos *<RopaSuperior>*, *<RopaInferior>* y *<Calzado>*, que se encuentran dentro de la etiqueta *<Modelizar>*. Los colores disponibles son:

- ✦ Rojo
- ✦ Blanco
- ✦ Azul
- ✦ Magenta
- ✦ Gris
- ✦ Verde
- ✦ Amarillo
- ✦ Cian
- ✦ Negro

En caso de indicar el color de la ropa incorrectamente, se tomará el color negro, por defecto, para todas las prendas.

A.4.8. Protagonistas

Los personajes que intervienen decisivamente en las historias se encuentran en el archivo `protagonistas.xml`. Es por ello que estos personajes son más conocidos como protagonistas.

A.4.8.1. Estructura básica del archivo

```
<?xml version="1.0" encoding="UTF-8" ?>
<ContenedorProtagonistas>
  <PersonajesPrincipales>
  </PersonajesPrincipales>
</ContenedorProtagonistas>
```

Dentro del elemento *<ContenedorProtagonistas>*, se situará la lista de personajes que aparecen en el juego usando la etiqueta *<PersonajesPrincipales>*.

A.4.8.2. Añadir un protagonista

```
<PersonajePrincipal id="pp1" idHistoria="Historia1" idObjetivo="Punto7"
    nombre="Miguel" camara="si" modelo="Prefap_ChicoDePie1" animado="si">
  <Posicion>
    <X>16.772</X> <Y>0.138</Y> <Z>11.038</Z>
  </Posicion>
  <Rotacion>
    <X>0</X> <Y>335.4413</Y> <Z>0</Z>
  </Rotacion>
```

```
    <Hablar>¿Dinero? Lo siento, tío. He traído lo justo para mi. No tengo
        más.</Hablar>
    <ObjetivosSeguir>
        <Posicion id="1">
            <X>16.772</X>
            <Y>0.7</Y>
            <Z>11.038</Z>
        </Posicion>
    </ObjetivosSeguir>
    <ModelizarFigura>
        <RopaSuperior>
            <R>255</R> <G>164</G> <B>32</B>
        </RopaSuperior>
        <RopaInferior>
            <R>30</R> <G>30</G> <B>30</B>
        </RopaInferior>
        <Calzado>
            <R>0</R> <G>0</G> <B>0</B>
        </Calzado>
    </ModelizarFigura>
</PersonajePrincipal>
```

Para añadir un protagonista se debe usar la etiqueta *<PersonajePrincipal>* y siete atributos: su identificador de personaje, el identificador de la historia a la que pertenece, el identificador del objetivo en el que aparecerá, su nombre, un atributo que indique si se desea usar una cámara para centrar la atención del usuario en el protagonista, el modelo del personaje que se desee utilizar y, por último, el atributo *animado* que indica si el personaje incorpora la animación que le permite caminar o estar en reposo. Los modelos disponibles de los protagonistas son los mismos que los de los personajes.

Además, se deberán especificar las coordenadas de la ubicación y rotación del personaje con las etiquetas *<Posicion>* y *<Rotacion>*, respectivamente. El mensaje que contienen los personajes y que se desvelará cuando el jugador se acerque a ellos se indicará con la etiqueta *<Hablar>* y el texto en su interior.

El protagonista es capaz de desplazarse hacia ciertos puntos de la facultad, tanto si se usa una cámara para mostrar el movimiento como si no. Es preferible que se use la animación del personaje al desplazarse para dar mayor realismo. Estos puntos deberán situarse dentro de la etiqueta *<Objetivos a seguir>*.

También se permite modificar los colores de la ropa del protagonista a través de los elementos *<RopaSuperior>*, *<RopaInferior>* y *<Calzado>*, que se encuentran dentro de la etiqueta *<ModelizarFigura>*. A diferencia de los personajes de relleno, éstos utilizan el modelo de color `RGB`, por lo que se especificará la cantidad de rojo, verde y azul con los elementos *<R>*, *<G>* y *<B>*, respectivamente.

A.4.8.3. Añadir un punto a seguir

Los puntos a seguir se encuentran en el interior de la etiqueta *<Objetivos-Seguir>*. Se añaden como elementos *<Posicion>* y su correspondiente identificador. Dentro de estos se encuentran las coordenadas X,Y y Z del punto exacto al que tiene que moverse el protagonista.

Bibliografía

Abad, R. C. *Introducción a la Simulación y a la Teoría de Colas*. Netbiblo, 2002.

Ballesteros, R. H. Expresiones de referencia y figuras retóricas para la distinción y descripción de entidades en discursos generados automáticamente. 2009. Disponible en `http://eprints.sim.ucm.es/9650/1/T31476.pdf` (último acceso, April, 2015).

Dutoit, T. *An Introduction to Text-to-Speech Synthesis*. Springer Science & Business Media, 2013.

Heileman, G. L. *Estructuras de datos, algoritmos y programación orientada a objetos*. McGraw-Hill, 1996.

Lebowitz, J. *Interactive Storytelling for Video Games*. Taylor & Francis, 2012.

Minkoff, J. The last of us. 2013. Disponible en `http://www.thelastofus.playstation.com` (último acceso, May, 2016).

Murphy, B. Orange juice font. 2005. Disponible en `http://www.dafont.com/es/orange-juice.font` (último acceso, May, 2016).

Planells, A. J. La evolución narrativa en los videojuegos de aventuras. 2010. Disponible en `http://www.ehu.eus/zer/hemeroteca/pdfs/zer29-06-planells.pdf` (último acceso, May, 2016).

Regueiro, L. I. El metajuego y la subversión de la narrativa tradicional en the stanley parable. 2015. Disponible en `https://periodicos.ufsc.br/index.php/textodigital/article/viewFile/1807-9288.2015v11n2p71/30955` (último acceso, February, 2015).

Salter, A. *What Is Your Quest?: From Adventure Games to Interactive Books*. University of Iowa Press, 2014.

TECHNOLOGIES, U. Unity 3d. 2005. Disponible en `https://unity3d.com/es` (último acceso, May, 2016).

VALVE. Half-life. 2004. Disponible en `http://www.valvesoftware.com/games/hl2.html` (último acceso, May, 2016).

WIKIPEDIA (Blender). Entrada: "Blender". Disponible en `https://es.wikipedia.org/wiki/Blender` (último acceso, Mayo, 2016).

WIKIPEDIA (Narrativa). Entrada: "Narrativa". Disponible en `https://es.wikipedia.org/wiki/Narrativa` (último acceso, Mayo, 2016).

WILLIAMS, R. King's quest. 1984. Disponible en `http://www.sierra.com/kingsquest` (último acceso, May, 2016).

WOLF, M. J. P. *Encyclopedia of Video Games: M-Z*. ABC-CLIO, 2012.